29. Theorietag
Automaten und Formale Sprachen

Bremen, 26./27. September 2019

Technischer Bericht der AG Datenbanken
Fachbereich 3
Universität Bremen
Herausgeber: Sebastian Maneth

Vorwort

Theorietage haben eine lange Tradition in verschiedenen Fachgruppen der Gesellschaft für Informatik (GI). Die Theorietage der GI Fachgruppe "Automaten und Formalen Sprachen" finden seit 1991 jährlich statt. Es ist auch Tradition, vor dem eigentlich Theorietag einen Workshop mit eingeladenen Vorträgen zu organisieren. Auf dem diesjährigen 29. Theorietag gibt es keinen gesonderten Workshop, sondern die eingeladenen Vorträge werden in das Gesammtprogramm eingeflochten. Dies führt dazu, dass die Länge des Theorietags anderthalb Tage anstatt einem Tag beträgt.

Zum diesjärigen Theorietag sind 37 Teilnehmer registriert und es werden 18 ordentliche Vorträge gehalten. Der vorliegende technische Bericht enthält die Zusammenfassungen dieser 18 Vorträge. Er wird elektronisch während des Theorietages zur Verfügung gestellt, damit Teilnehmer dort bei Bedarf die vorgestellten Resultate nachschlagen können.

Ausserdem werden drei eingeladenen Vorträge gehalten. Das übergeordnete Thema der eingeladenen Vorträge ist "Datenbanktheorie". Die folgenden herausragende Wissenschaftler konnten eingeladen werden und halten Voträge zu folgenden Themen:

- Mikołaj Bojańczik (Universität Warschau, Polen)
  Polyregular Functions
- Wim Martens (Universität Bayreuth)
  Optimization and Evaluation of Real-Life Graph Queries
- Thomas Schwentick (Technische Universität Dortmund)
  Dynamic Complexity: Recent and Complex Updates

Wir danken der Gesellschaft für Informatik für die freundliche Unterstützung, die es ermöglicht, einen Teil der Reisekosten der eingeladenen Vortragenden abzudecken. Wir danken der Universität Bremen für finanzielle Unterstützung und für die Bereitstellung der Räume.

Wir wünschen allen Teilnehmenden interessante und anregende Theorietage in Bremen!

Sebastian Maneth
Bremen, 22.09.2019

Conference Program

THEORIE-AG 2019 S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 1–1.
Universität Bremen

On Solution Sets of Word Equations

Dirk Nowotka

Dependable Systems Group, Dept. of Computer Science,
Kiel University, 24098 Kiel, Germany
dn@zs.uni-kiel.de

Abstract

We take a fresh look at word equations which is different from the approaches of Makanin (exponent of periodicity) and Jez (recompression). Using representations of letters as arbitrary but distinct numbers, we are able to find some normalisations on the representation of the solutions of word equations which enable us to solve two long-standing open problems. Firstly, we prove that a one variable word equation with constants has either at most three or an infinite number of solutions. This bound is both surprising and optimal. Secondly, we consider independent systems of three variable word equations without constants and establish a constant bound on the size of such systems, if they allow a nonperiodic solution.

THEORIE-AG 2019 S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 2–2.
Universität Bremen

On Iterated Uniform Finite-State Transducers

Martin Kutrib[(A)] Andreas Malcher[(A)] Carlo Mereghetti[(B)]
Beatrice Palano[(C)]

[(A)]Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
kutrib@informatik.uni-giessen.de
andreas.malcher@informatik.uni-giessen.de

[(B)]Dipartimento di Fisica "Aldo Pontremoli",
Università degli Studi di Milano
via Celoria 16, 20133 Milano, Italy
mereghetti@unimi.it

[(C)]Dipartimento di Informatica "G. degli Antoni",
Università degli Studi di Milano
via Celoria 18, 20133 Milano, Italy
palano@unimi.it

Zusammenfassung

We introduce the deterministic computational model of an *iterated uniform finite-state transducer (*IUFST*)*. A IUFST performs the same length-preserving transduction on several left-to-right sweeps. The first sweep takes place on the input string, while any other sweep processes the output of the previous one. The IUFST accepts or rejects upon halting in an accepting or rejecting state along its sweeps. First, we focus on constant sweep bounded IUFSTs. We study their descriptional power vs. deterministic finite automata, and the state cost of implementing language operations. Then, we focus on non-constant sweep bounded IUFSTs, showing a nonregular language hierarchy depending on the sweep complexity. The hardness of some classical decision problems on constant sweep bounded IUFSTs is also investigated.

THEORIE-AG 2019 S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 3–6.
Universität Bremen

Semirecognizable Sets and Right One-Way Jumping Finite Automata

Simon Beier Markus Holzer

Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{simon.beier,holzer}@informatik.uni-giessen.de

Abstract

We generalize the notion of recognizable subsets of monoids and call a subset of a monoid semirecognizable if it is the premimage of a finite set under a monoid homomorphism. Especially semirecognizable subsets of the free commutative moniod $\mathbb{N}^k$ are investigated by us; those build a subfamily of the semilinear sets. A key role in our theory is played by lattices, which are special semirecognizable sets that extend "the pattern" of a linear set to the whole $\mathbb{N}^k$. We show connections between semirecognizable subsets of $\mathbb{N}^k$ and rational cones. Right one-way jumping finite automata (ROWJFAs) are an automaton model for deterministic but non-contiguous information processing that was introduced in 2016 by Chigahara, Fazekas, and Yamamura. In 2018 we showed that the permutation closed languages accepted by ROWJFAs are exactly the permutation closed semirecognizable languages. Using our results about semirecognizable subsets of $\mathbb{N}^k$ we give a characterization of permutation closed languages accepted by ROWJFAs with multiple initial states.

1. Introduction

A subset S of a monoid M is called *recognizable* if there are a finite monoid N and a monoid homomorphism $f : M \to N$ such that $S = f^{-1}(f(S))$. So, for each $a \in M$ the value $f(a)$ tells us if $a \in S$. There is a connection between recognizability and the syntactic congruence, which was introduced in a paper by Rabin and Scott and was credited to Myhill [7]. Given a subset S of a monoid M, the *syntactic congruence* $\sim_S$ is an equivalence relation on M. For $a, b \in M$ we have $a \sim_S b$ if for all $c, d \in M$ the condition $cad \in S$ is equivalent to the condition $cbd \in S$. The set $M/\sim_S$ is a monoid, called the *syntactic monoid* of S. A subset of a monoid is recognizable if and only if its syntactic monoid is finite. The recognizable subsets of the free monoid over a finite set are the regular languages. *McKnight's Theorem* says that each recognizable subset of a finitely generated monoid is rational. This gives us especially that each recognizable subset of the free commutative monoid $\mathbb{N}^k$ is semilinear. A permutation closed language is regular if and only if its Parikh-image is a recognizable subset of $\mathbb{N}^k$. The recognizable subsets of $\mathbb{N}^k$ are well understood, because *Mezei's Theorem* states that these are exactly the finite unions of direct products of semilinear subsets of $\mathbb{N}$. Using Presburger arithmetic, Ginsburg and Spanier showed that it is deciadable if a given semilinear set is recognizable [6].

Recognizability is generalized by us in the following way. A subset S of a monoid M is called *semirecognizable* if there are a monoid N and a monoid homomorphism $f : M \to N$

such that $f(S)$ is finite and $S = f^{-1}(f(S))$. This is equivalent to the condition that the projection $S/\sim_S$ of S to its syntactic monoid is finite. If we replace "finite" by "a singleton or the empty set" in the last two sentences, we get the notion of a *strongly semirecognizable* subset of a monoid. We are especially interested in (strongly) semirecognizable subsets of $\mathbb{N}^k$. Each semirecognizable subset of $\mathbb{N}^k$ is a finite union of strongly semirecognizable linear subsets of $\mathbb{N}^k$. It is decidable whether a given semilinear set is (strongly) semirecognizable. We characterize when a linear subset of $\mathbb{N}^k$ is semirecognizable and show a connection to rational cones. Lattices are introduced as special strongly semirecognizable subsets of $\mathbb{N}^k$. They are defined like linear subsets of $\mathbb{N}^k$, but allowing integer coefficients for the period vectors, instead of only natural numbers. However our lattices are still, per definition, subsets of $\mathbb{N}^k$. Each strongly semirecognizable subset of $\mathbb{N}^k$ equals a lattice if we only consider vectors where all components are "large enough". Motivated by the result of Ginsburg and Spanier that each semilinear set is a finite union of linear sets with linearly independent periods [5], we study in which cases lattices and arbitrary strongly semirecognizable subsets of $\mathbb{N}^k$ are a finite union of (strongly) semirecognizable linear sets with linearly independent periods; again there is a connection to rational cones. That is why we study these objects in more detail and show that the set of vectors with only non-negative components in a linear subspace of dimension n of $\mathbb{R}^k$ spanned by a subset of $\mathbb{N}^k$ always forms a rational cone spanned by a linearly independent subset of $\mathbb{N}^k$ if and only if $n \in \{0, 1, 2, k\}$. A result is given that states in terms of lattices when an arbitrary subset of $\mathbb{N}^k$ is a finite union of (strongly) semirecognizable (linear) subsets of $\mathbb{N}^k$.

Right one-way jumping finite automata (ROWJFAs) were proposed by Chigahara, Fazekas, and Yamamura [4]. These automata are defined like DFAs with a partial transition function, but process the input differently: If the device cannot read the current symbol, the head jumps over it and continues the computation with the next one. When the head reaches the end of the input, it jumps back to the beginning and goes on with its deterministic computation. We proved that the permutation closed languages accepted by ROWJFAs are exactly the permutation closed semirecognizable languages [1]. A characterization of permutation closed languages accepted by ROWJFAs with multiple initial states is given by applying our results about semirecognizable subsets of $\mathbb{N}^k$. The results of the current paper are mainly from [3], see also [2].

2. Results

A subset of $\mathbb{N}^k$ is called *linear* if it is of the form $\mathsf{L}(\vec{c}, P) = \{\, \vec{c} + \sum_{\vec{p} \in P} \lambda_{\vec{p}} \cdot \vec{p} \mid \forall \vec{p} \in P : \lambda_{\vec{p}} \in \mathbb{N} \,\}$ for a finite set $P \subseteq \mathbb{N}^k$ and a $\vec{c} \in \mathbb{N}^k$, where P is called the set of *periods* and $\vec{c}$ is called the *constant vector* of this representation. A subset of $\mathbb{N}^k$ is called *semilinear* if it is a finite union of linear subsets of $\mathbb{N}^k$. Clearly, the (strongly) semirecognizability of a linear subset of $\mathbb{N}^k$ depends only on its period set, but not on the constant vector.

Proposition 2.1 *Each semirecognizable subset of $\mathbb{N}^k$ is a finite union of strongly semirecognizable linear subsets of $\mathbb{N}^k$. It is decidable if a given semilinear set is (strongly) semirecognizable.*

For a set $S \subseteq \mathbb{R}^k$ let $\mathsf{span}(S)$ be the linear subspace of $\mathbb{R}^k$ spanned by S. For a finite $S \subseteq \mathbb{Z}^k$ the *rational cone spanned by* S is $\mathsf{cone}(S) = \{\, \sum_{\vec{s} \in S} \lambda_{\vec{s}} \cdot \vec{s} \mid \forall \vec{s} \in S : \lambda_{\vec{s}} \in \mathbb{R}_{\geq 0} \,\} \subseteq \mathbb{R}^k$. A *linearly independent rational cone* in $\mathbb{R}^k$ is a set of the form $\mathsf{cone}(S)$ for a linearly independent $S \subseteq \mathbb{Z}^k$. It is well known that each rational cone is a finite union of linearly independent rational cones.

We define two properties of subsets of $\mathbb{N}^k$ which involve rational cones. Let $k \geq 0$ and $S \subseteq \mathbb{N}^k$. Then, the set S has the *linearly independent rational cone property* if $\mathsf{span}(S) \cap (\mathbb{R}_{\geq 0})^k = \mathsf{cone}(T)$, for some linearly independent $T \subseteq \mathbb{N}^k$. The set S has the *own rational cone property* if S is finite and it holds $\mathsf{span}(S) \cap (\mathbb{R}_{\geq 0})^k = \mathsf{cone}(S)$.

Theorem 2.2 *Let $k \geq 0$ and $P \subseteq \mathbb{N}^k$ be finite. Then, $\mathsf{L}(\vec{0}, P)$ is semirecognizable if and only if P has the own rational cone property. If P is linearly independent, $\mathsf{L}(\vec{0}, P)$ is semirecognizable if and only if it is strongly semirecognizable.*

A subset of $\mathbb{N}^k$ is called a *lattice* if it is of the form $\mathsf{La}(\vec{c}, P) = \{ \vec{c} + \sum_{\vec{p} \in P} \lambda_{\vec{p}} \cdot \vec{p} \mid \forall \vec{p} \in P : \lambda_{\vec{p}} \in \mathbb{Z} \} \cap \mathbb{N}^k$ for a finite set $P \subseteq \mathbb{N}^k$ and a $\vec{c} \in \mathbb{N}^k$. Clearly, each lattice is a strongly semirecognizable subset of $\mathbb{N}^k$. For $\vec{x}, \vec{y} \in \mathbb{R}^k$ we write $\vec{x} \leq \vec{y}$ if all components of $\vec{x}$ are less or equal to the corresponding components of $\vec{y}$.

Lemma 2.3 *Let $k \geq 0$, S be a strongly semirecognizable subset of $\mathbb{N}^k$, and $\vec{s} \in S$. Then, there is a lattice $L \subseteq \mathbb{N}^k$ such that $\{ \vec{x} \in S \mid \vec{x} \geq \vec{s} \} = \{ \vec{x} \in L \mid \vec{x} \geq \vec{s} \}$.*

Theorem 2.4 *For $k \geq 0$, $\vec{c} \in \mathbb{N}^k$, and a finite $P \subseteq \mathbb{N}^k$ the following conditions are equivalent: 1. The lattice $\mathsf{La}(\vec{c}, P)$ is a finite union of semirecognizable linear sets with linearly independent periods. 2. The lattice $\mathsf{La}(\vec{c}, P)$ is a finite union of strongly semirecognizable linear sets with linearly independent periods. 3. The set P has the linearly independent rational cone property.*

There is a generalization of Theorem 2.4 from lattices to strongly semirecognizable sets.

Theorem 2.5 *Let $k \geq 0$ and $n \in \{0, 1, \ldots, k\}$. Then, the condition that each $S \subseteq \mathbb{N}^k$ fulfilling $\dim(\mathsf{span}(S)) = n$ has the linearly independent rational cone property holds if and only if $n \in \{0, 1, 2, k\}$.*

A subset $S \subseteq \mathbb{N}^k$ is called a *quasi lattice* if there are $\vec{y} \in \mathbb{N}^k$, $m \geq 0$, $\vec{c_1}, \vec{c_2}, \ldots, \vec{c_m} \in \mathbb{N}^k$, and finite $P_1, P_2, \ldots, P_m \subseteq \mathbb{N}^k$ such that the set $\{ \vec{x} \in S \mid \vec{x} \geq \vec{y} \}$ is equal to $\{ \vec{x} \in \bigcup_{j=1}^{m} \mathsf{La}(\vec{c_j}, P_j) \mid \vec{x} \geq \vec{y} \}$. For $k \geq 0$ and $T \subseteq \{1, 2, \ldots, k\}$ with $T = \{t_1, t_2, \ldots, t_{|T|}\}$ and $t_1 < t_2 < \cdots < t_{|T|}$ we define $\pi_{k,T} : \mathbb{N}^k \to \mathbb{N}^{|T|}$ through $\pi_{k,T}(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_k) = \left(\vec{x}_{t_1}, \vec{x}_{t_2}, \ldots, \vec{x}_{t_{|T|}} \right)$.

Theorem 2.6 *For a $k \geq 0$ and a subset $S \subseteq \mathbb{N}^k$ the following three conditions are equivalent: 1. The set S is a finite union of semirecognizable sets. 2. The set S is a finite union of strongly semirecognizable linear sets. 3. For all $T \subseteq \{1, 2, \ldots, k\}$ and $\vec{x} \in \mathbb{N}^{|T|}$ the set $\pi_{k,\{1,2,\ldots,k\} \setminus T} \left(\{ \vec{z} \in S \mid \pi_{k,T}(\vec{z}) = \vec{x} \} \right)$ is a quasi lattice.*

A ROWJFA with multiple initial states is a system $A = (Q, \Sigma, \delta, S, F)$, where the symbols have the same meaning as for an ordinary DFA with multiple initial states and a partial transition function. We define the binary relation $\circlearrowright_A$ on the set $Q\Sigma^*$ as follows. For $p, q \in Q$, $a \in \Sigma$, and $w \in \Sigma^*$ with $\delta(p, a) = q$ we have $paw \circlearrowright_A qw$. On the other hand, for $p \in Q$, $a \in \Sigma$, and $w \in \Sigma^*$ such that $\delta(p, a)$ is undefined we have $paw \circlearrowright_A pwa$. The language accepted by A is $L(A) = \{ w \in \Sigma^* \mid \exists s \in S, f \in F : sw \circlearrowright_A^* f \}$. Clearly, each language accepted by a ROWJFA with multiple initial states is a finite union of languages accepted by ROWJFAs with a single initial state. However, it is not clear that each permutation closed language accepted by a ROWJFA with multiple initial states is a finite union of permutation closed languages accepted

by ROWJFAs with a single initial state. Such a characterization would be very useful, because in an earlier paper we have shown that the permutation closed languages accepted by ROWJFAs with a single initial state are exactly the permutation closed semirecogniazable languages [1].

Theorem 2.7 *The Parikh-image of each permutation closed language accepted by a ROWJFA with multiple initial states is a quasi lattice.*

To get a characterization result, we define the language operation of *disjoint quotient* of a language $L \subseteq \Sigma^*$ with a word $w \in \Sigma^*$ as follows:

$$L/^d w = \{\, v \in \Sigma^* \mid vw \in L, \forall a \in \Sigma : |v|_a \cdot |w|_a = 0 \,\} = (L/w) \cap \{\, a \in \Sigma \mid |w|_a = 0 \,\}^*.$$

The family of permutation closed semirecogniazable languages is closed under quotient with a word and under disjoint quotient with a word. For an alphabet Σ, $\Gamma \subseteq \Sigma$, and an $L \subseteq \Sigma^*$ accepted by a ROWJFA with multiple initial states, $L \cap \Gamma^*$ is also accepted by such a device. Using Theorem 2.7 and our results about semirecognizable subsets of $\mathbb{N}^k$, we get:

Corollary 2.8 *Let Σ be an alphabet and $L \subseteq \Sigma^*$ be permutation closed. Then, L is a finite union of permutation closed semirecognizable languages if and only if for all $w \in \Sigma^*$ the language $L/^d w$ is accepted by a ROWJFA with multiple initial states.*

Notice that for binary alphabets the last condition is equivalent to the fact that L is accepted by a ROWJFA with multiple initial states.

References

[1] S. BEIER, M. HOLZER, Properties of Right One-Way Jumping Finite Automata. In: S. KONSTANTINIDIS, G. PIGHIZZINI (eds.), *Proceedings of the* 20*th International Workshop on Descriptional Complexity of Formal Systems*. Number 10952 in LNCS, Springer, Halifax, Nova Scotia, Canada, 2018, 11–23.

[2] S. BEIER, M. HOLZER, *Semi-Linear Lattices and Right One-Way Jumping Finite Automata*. IFIG Research Report 1901, Institut für Informatik, Justus-Liebig-Universität Gießen, Arndtstr. 2, D-35392 Gießen, Germany, 2019.

[3] S. BEIER, M. HOLZER, Semi-Linear Lattices and Right One-Way Jumping Finite Automata (Extended Abstract). In: M. HOSPODÁR, G. JIRÁSKOVÁ (eds.), *Proceedings of the* 24*th International Conference on Implementation and Application of Automata*. LNCS, Springer, Košice, Slovakia, 2019, 70–82.

[4] H. CHIGAHARA, S. FAZEKAS, A. YAMAMURA, One-Way Jumping Finite Automata. *Internat. J. Found. Comput. Sci.* **27** (2016), 391–405.

[5] S. GINSBURG, E. H. SPANIER, Bounded ALGOL-like languages. *Trans. AMS* **113** (1964), 333–368.

[6] S. GINSBURG, E. H. SPANIER, Bounded Regular Sets. *Proc. Amer. Math. Soc.* **17** (1966), 1043–1049.

[7] M. O. RABIN, D. SCOTT, Finite Automata and Their Decision Problems. *IBM Journal of Research and Development* **3** (1959), 114–125.

THEORIE-AG 2019 S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 7–7.
Universität Bremen

Tree Substitution Grammars

Andreas Maletti

Fakultät für Mathematik und Informatik, Universität Leipzig
PO box 100 920, 04009 Leipzig, Germany
maletti@informatik.uni-leipzig.de

Abstract

We report work-in-progress on the expressive power of tree substitution grammars, which are popular grammar mechanisms in the area of parsing of natural languages. A tree substitution grammar is essentially a finite set of tree fragments together with a set of initial labels. Fragments with root label σ can be attached to any tree with a leaf labeled σ by replacing the leaf directly by the whole fragment. Repeated replacements starting from a leaf labeled with an initial label yield the trees generated by the grammar and a tree language is a tree substitution language if there exists a tree substitution grammar, which generates it. It is well known that each local tree language is a tree substitution language and in turn each tree substitution language is regular.

In this contribution we show that tree substitution languages enjoy none of the standard Boolean closures, but at least all finite and co-finite tree languages are tree substitution languages. In addition, we report on progress on the characterization of tree substitution languages inside the regular tree languages.

THEORIE-AG 2019

S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 8–10.

Universität Bremen

Average Case Analysis of Leaf-Centric Binary Tree Sources

Louisa Seelbach Benkner[(A)] Markus Lohrey[(A)]

[(A)]University of Siegen
Department of Electrical Engineering and Computer Science
{seelbach, lohrey}@eti.uni-siegen.de

Abstract

We study the average size of the minimal directed acyclic graph (DAG) with respect to so-called leaf-centric binary tree sources as studied by Zhang, Yang, and Kieffer in [12]. A leaf-centric binary tree source induces for every $n \geq 2$ a probability distribution on all binary trees with n leaves. We generalize a result shown by Flajolet, Gourdon, Martinez [6] and Devroye [5] according to which the average size of the minimal DAG of a binary tree that is produced by the binary search tree model is $\Theta(n/\log n)$, and a result shown by Flajolet, Sipala and Steyaert [7], according to which the average size of the minimal DAG of a binary tree with respect to the uniform probability distribution on all trees of size n is $\mathcal{O}(n/\sqrt{\log n})$.

Introduction

One of the most important and widely used compression methods for trees is to represent a tree by its minimal *directed acyclic graph*, shortly referred to as minimal DAG. The minimal DAG of a tree t is obtained by keeping for each subtree s of t only one isomorphic copy of s to which all edges leading to roots of s-copies are redirected. DAGs found applications in numerous areas of computer science; let us mention compiler construction [1, Chapter 6.1 and 8.5], unification [10], XML compression and querying [4, 8], and symbolic model-checking (binary decision diagrams) [3].

We consider the problem of derving asymptotic estimates for the average size of the minimal DAG of a randomly chosen binary tree of size n. So far, this problem has been analyzed mainly for two particular distributions: In [7], Flajolet, Sipala and Steyaert proved that the average size of the minimal DAG with respect to the uniform distribution on all binary trees of size n is asymptotically equal to $c \cdot n/\sqrt{\ln n}$, where c is the constant $2\sqrt{\ln(4/\pi)}$. This result was extended to unranked and node-labelled trees in [2] (with a different constant c). An alternative proof to the result of Flajolet et al. was presented in [11] by Ralaivaosaona and Wagner. For the so-called binary search tree model, Flajolet, Gourdon and Martinez [6] and Devroye [5] proved that the average size of the minimal DAG becomes $\Theta(n/\log n)$. In the binary search

[(A)]This work has been supported by the DFG research project LO 748/10-1 (QUANT-KOMP)

tree model, a binary search tree of size n is built by inserting the keys $1, \ldots, n$ according to a uniformly chosen random permutation on $1, \ldots, n$.

A general concept to produce probability distributions on the set of binary trees of size n was introduced by Zhang, Yang, and Kieffer in [12] (see also [9]), where the authors introduce so-called *leaf-centric binary tree sources*. This yields a general framework for studying the average size of a minimal DAG. Both the uniform distribution on all trees with n leaves and the binary search tree model can be modeled as a leaf-centric binary tree source.

We generalize the results of [5, 6] and [7] on the average size of the minimal DAG with respect to the binary search tree model, respectively, the uniform distribution, by considering several classes of leaf-centric binary tree sources, for which we obtain asymptotic bounds for the average size of the minimal DAG.

References

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science / World student series edition. Addison-Wesley, 1986.

[2] Mireille Bousquet-Mélou, Markus Lohrey, Sebastian Maneth, and Eric Noeth. XML compression via DAGs. *Theory of Computing Systems*, 57(4):1322–1371, 2015.

[3] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[4] Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed XML. In Johann Christoph Freytag et al., editors, *Proceedings of the 29th Conference on Very Large Data Bases (VLDB 2003)*, pages 141–152. Morgan Kaufmann, 2003.

[5] Luc Devroye. On the richness of the collection of subtrees in random binary search trees. *Inf. Process. Lett.*, 65(4):195–199, 1998.

[6] Philippe Flajolet, Xavier Gourdon, and Conrado Martínez. Patterns in random binary search trees. *Random Struct. Algorithms*, 11(3):223–244, 1997.

[7] Philippe Flajolet, Paolo Sipala, and Jean-Marc Steyaert. Analytic variations on the common subexpression problem. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP 1990)*, volume 443 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 1990.

[8] Markus Frick, Martin Grohe, and Christoph Koch. Query evaluation on compressed trees (extended abstract). In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'2003)*, pages 188–197. IEEE Computer Society Press, 2003.

[9] John C. Kieffer, En-Hui Yang, and Wojciech Szpankowski. Structural complexity of random binary trees. In *IEEE International Symposium on Information Theory, ISIT 2009*, pages 635–639. IEEE, 2009.

[10] Mike Paterson and Mark N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16(2):158–167, 1978.

[11] Dimbinaina Ralaivaosaona and Stephan G. Wagner. Repeated fringe subtrees in random rooted trees. In *Proceedings of the Twelfth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2015*, pages 78–88. SIAM, 2015.

[12] Jie Zhang, En-Hui Yang, and John C. Kieffer. A universal grammar-based code for lossless compression of binary trees. *IEEE Transactions on Information Theory*, 60(3):1373–1386, 2014.

THEORIE-AG 2019

S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 11–15.

Universität Bremen

Generating Hypergraph Languages by (Context-dependent) Fusion Grammars and Splitting/Fusion Grammars

Hans-Jörg Kreowski Sabine Kuske Aaron Lye

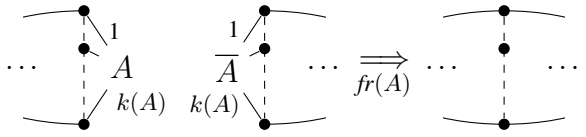
University of Bremen, Department of Computer Science
P.O.Box 33 04 40, 28334 Bremen, Germany
{kreo,kuske,lye}@informatik.uni-bremen.de

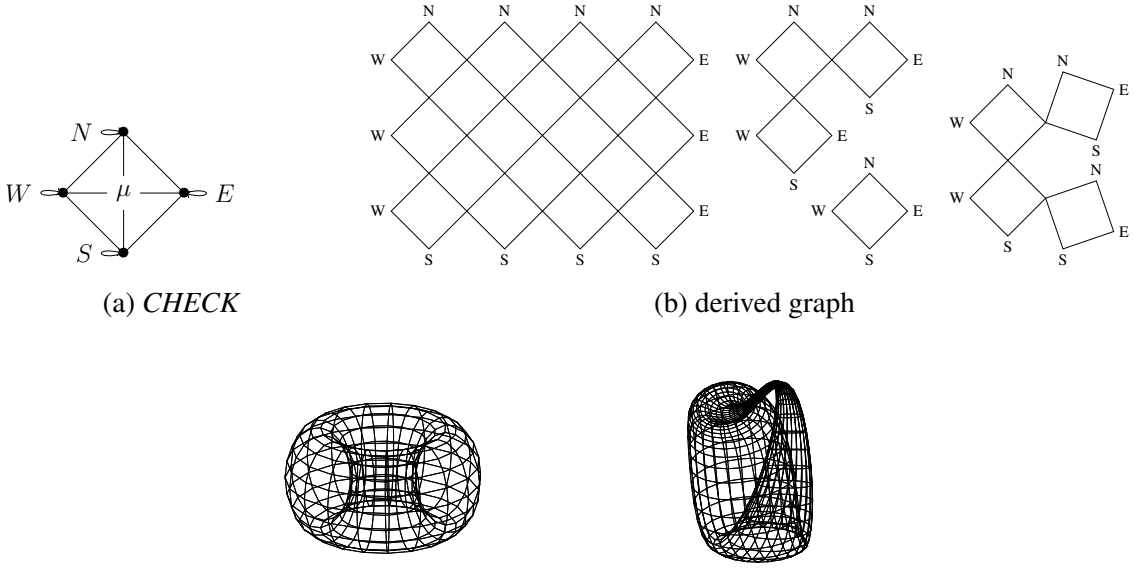
Extended Abstract

Fusion grammars, context-dependent fusion grammars and splitting/fusion grammars were recently introduced as devices for the generation of hypergraph languages (see [**?**, **?**, **?**]). They are inspired by basic operations of DNA computing (cf, e.g., [**?**]). A fusion grammar provides a start hypergraph and a finite set of fusion labels (besides some markers and terminals). The fusion labels have complements and serve as rules. A fusion is defined by choosing two complementarily labeled hyperedges, removing them and merging the corresponding attachment vertices. Given a hypergraph, the set of all possible fusions is finite as fusions never create anything. To overcome this limitation, we allow arbitrary multiplications of connected components, i.e., connected subhypergraphs of maximal size, within derivations in addition to fusion. A useful generalization is the restriction of fusions by positive and negative context-conditions. Splitting is the inverse operation to fusion, i.e., a sequence of vertices is chosen, each of these may be split in two vertices and two complementarily labeled hyperedges are attached. In this presentation, we survey the very first results on these grammars.

Let Σ be a finite alphabet. A *hypergraph* over Σ is a system $H = (V, E, att, lab)$ where V is a finite set of *vertices*, E is a finite set of *hyperedges*, $att\colon E \to V^*$ is a function, called *attachment* (assigning a string of attachment vertices to each edge), and $lab\colon E \to \Sigma$ is a function, called *labeling*. If some label $x \in \Sigma$ has got a type $k(x)$, then the length of the attachment of each hyperedge labeled with x has length $k(x)$. $\mathcal{H}_\Sigma$ denotes the set of all hypergraphs labeled over Σ.

A *fusion grammar* is a system $FG = (Z, F, M, T)$ where Z is a finite *start hypergraph* and F, M and T are pairwise-disjoint subalphabets of Σ. Their elements are called *fusion labels*, *markers*, and *terminals*, respectively. Each $A \in F$ has a type $k(A) \in \mathbb{N}$ and a complement $\overline{A}$ of the same type. $A \in F$ represents a *fusion rule* $fr(A)$ which is applied to $H \in \mathcal{H}_\Sigma$ by (1) choosing two complementarily labeled hyperedges, (2) removing them, and (3) merging the corresponding attachment vertices yielding the hypergraph H':





(a) *CHECK* (b) derived graph

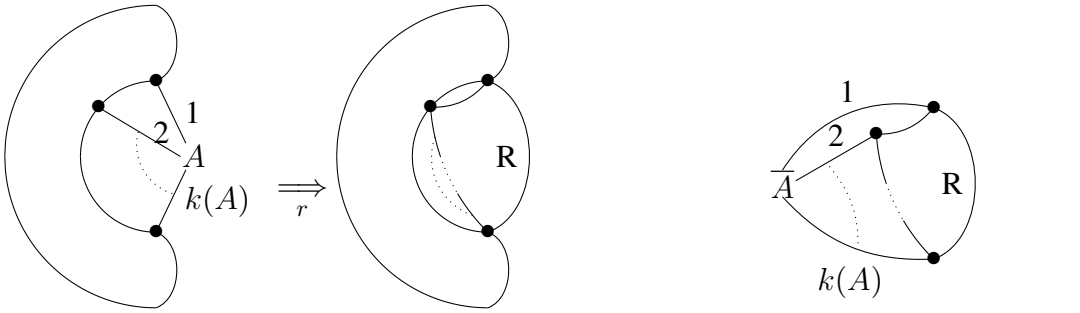
(c) torus (d) Klein bottle

Figure 1:: Pseudotori

This is denoted by $H \underset{fr(A)}{\Longrightarrow} H'$. Besides such a rule application, a direct derivation may be a multiplication $H \underset{m}{\Longrightarrow} m \cdot H$ for some multiplicity m which assigns a non-negative integer to each connected component C of H. The result $m \cdot H$ consists of the disjoint union of $m(C)$ copies of C for each connected component C of H. A *derivation* $H \overset{n}{\Longrightarrow} H'$ of length n is a sequence $H_0 \Longrightarrow H_1 \Longrightarrow \ldots \Longrightarrow H_n$ with $H = H_0$ and $H' = H_n$ including the case $n = 0$. One may write $H \overset{*}{\Longrightarrow} H'$ for arbitrary n. The *generated language* of *FG* is $L(\mathit{FG}) = \{rem_M(H) \mid Z \overset{*}{\Longrightarrow} H, H \in \mathcal{H}_{T \cup M} - \mathcal{H}_T, H \text{ connected}\}$ where $rem_M(H)$ is the hypergraph obtained when removing all hyperedges with labels in M.

Example Consider the set $\{N, W, S, E\}$ and let $F = \{N, W\}$ with $k(N) = k(W) = 1$ and $\overline{N} = S, \overline{W} = E$. Then the fusion grammar $\mathit{PSEUDOTORI} = (\mathit{CHECK}, F, \{\mu\}, \{*\})$ with *CHECK* depicted in Fig. 1a generates graphs of structures related to tori and Klein bottles, as the following reasoning indicates.

Starting with a multiplication by – say – 20, the fusion of disjoint components only yields grid structures like the ones depicted in Fig. 1b where the markers and loops are omitted. If one continues now with fusions within connected components as long as possible, then one gets the resulting connected components as members of the generated language. Consider particularly the first connected component in Fig. 1b. If one fuses the vertices with W- and E-loops from top to bottom and the vertices with N- and S-loops from left to right, then one gets a torus (Fig. 1c, for simplicity the checks are replaced by rectangles). If one fuses the vertices with W- and E-loops in opposite order, then one gets a Moebius strip. The further fusion of the vertices with N- and S-loops from left to right yields the Klein bottle (Fig. 1d). All other terminal structures where the fusion is done in arbitrary order result in something between torus and Klein bottle. Hence we call them pseudotori.



(a) application of a hyperedge replacement rule r (b) fusion component of the rule r

Figure 2:: Components in the proof of Theorem 1

Fusion grammars are closely related to hyperedge replacement grammars [**?**, **?**]. A hyperedge replacement grammar is a system $\mathit{HRG} = (N, T, P, S)$ where N and T are disjoint subalphabets of Σ (elements in N and T are called *nonterminals* and *terminals*, respectively), $S \in N$ is the *start symbol* of type 0, and P is a finite set of *rules* where a rule is a triple $r = (A, R, ext)$ with $A \in N, R \in \mathcal{H}_\Sigma$, and ext is a sequence of $k(A)$ vertices of R. To apply r to some $H \in \mathcal{H}_\Sigma$, one chooses an A-hyperedge of H, replaces it by R, and merges corresponding attachment vertices and the ext-vertices as depicted in Fig. 2a. The language generated by HRG contains all terminal hypergraphs that can be derived from an S-labeled hyperedge by a sequence of rule applications. It is easy to see that a hyperedge replacement can be obtained by a fusion of the replaced hyperedge and the replacing right-hand side of the applied rule if one adds a hyperedge with complementary label to the right-hand side with ext as attachment as depicted in Fig. 2b. This observation leads to the following result shown in [**?**].

Theorem 1 Let *HRG* be a hyperedge replacement grammar with connected right-hand sides of rules and *FG*(*HRG*) be the corresponding fusion grammar. Then $L(\mathit{HRG}) = L(\mathit{FG}(\mathit{HRG}))$.

The converse does not hold. While the graph languages generated by hyperedge replacement grammars have bounded tree width, the set of square grids has unbounded tree width (cf. [**?**]) and consequently the set of pseudotori, too. This proves the following theorem.

Theorem 2 Fusion grammars are more powerful than hyperedge replacement grammars.

Context-dependent fusion grammars were introduced in [**?**] to simulate Petri nets. A context-dependent fusion rule is a triple (A, PC, NC) where A is a fusion symbol and PC, NC are – slightly sloppily seen – two sets of hypergraphs, called positive and negative context conditions, respectively. Such a rule is applicable like $fr(A)$, but only if both the positive and negative context conditions hold in the usual way of context conditions (adapted to hypergraphs). A context-dependent fusion grammar extends a fusion grammar by a finite set of context-dependent fusion rules. A derivation step is either a context-dependent fusion rule application or a multiplication. The generated language is defined as for ordinary fusion grammars.

In [**?**] it is shown that Turing machines can be simulated by context-dependent fusion grammars. The construction works roughly as follows: (1) A Turing machine is represented by the usual state graph, (2) the tape is represented by a sequence of successive edges each labeled

with symbols of the working alphabet, (3) the state graph and the tape are connected by a hyperedge that is attached to the current position on the tape and indicates the current state, (4) in addition, the start hypergraph contains components that allow to generate the initial tape in a terminal and a fusion version, and (5) components that allow to simulate a transition step of the Turing machine by a sequence of applications of context-dependent fusion rules. Finally, there is a terminating component that allows to disconnect the terminal tape with the input string from the rest of the working hypergraph whenever an accepting state is reached. In other words, the grammar generates a tape with an input string if and only if the Turing machine accepts this string. This yields the following result.

Theorem 3 Let *TM* be a Turing machine and $CDFG(TM)$ the corresponding context-dependent fusion grammar. Then $L(TM)$ and $L(CDFG(TM))$ coincide up to representation.

In [**?**] we defined the notion of splitting which is inverse to fusion, i.e., each of a chosen sequence of k vertices may be split into two vertices and equipped with a pair of complementarily labeled hyperedges of type k, but using a splitting complement instead of the fusion complement. A splitting rule (here with fixed disjoint context) consists of a fusion label and a context hypergraph. The application requires that the context becomes a disjoint component by splitting equipped with the complementary hyperedge. A splitting/fusion grammar is a fusion grammar extended by a finite set of splitting rules. A direct derivation is then either a fusion rule application, a splitting rule application or a multiplication. The generated language is defined as in the case of fusion grammars.

It can be shown that Chomsky grammars can be simulated by splitting/fusion grammars. The construction works roughly as follows: (1) Strings are represented by sequences of successive edges labeled by the symbols in the string from left to right, and (2) the start hypergraph consists of the graph of the initial string and the graphs of the right-hand sides of the Chomsky rules equipped with an extra edge that is labeled with the complement of the rule considered as fusion label. The splitting rules correspond to the Chomsky rules and have the left-hand sides as contexts. A spitting wrt a Chomsky rule r followed by the fusion wrt to r has the same effect on a string graph as the application of r to the string. This yields the following result.

Theorem 4 Let *CG* be a Chomsky grammar and $SFG(CG)$ the corresponding splitting/fusion grammar with rules restricted to fixed disjoint context. Then $L(CG)$ and $L(SFG(CG))$ coincide up to representation.

Moreover, we proved in [**?**] that connective hypergraph grammars can be simulated by these kind of grammars.

Summarizing, fusion grammars have turned out to be more powerful than hyperedge replacement grammars and the generalizations to context-dependent fusion grammars and splitting/fusion grammars are both universal in the sense that they cover the class of recursively enumerable languages. But the research on these type of hypergraph grammars is still in an early stage of development, and there are many open problems including:

1. Are fusion grammars, as we assume, not universal? And if so, is the emptiness problem or the membership problem or any other such problem decidable?

2. Are context-dependent fusion grammars restricted to either rules with positive context-conditions or negative context conditions still universal?

3. Is there a natural transformation from context-dependent fusion grammars into splitting/fusion grammars or the other way round?

THEORIE-AG 2019 S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 16–21.
Universität Bremen

Structural Sparsity

Sebastian Siebertz

University of Bremen

Abstract

Sparse graph classes, such as classes of bounded treewidth, classes of bounded expansion and nowhere dense classes, have a rich structural and algorithmic theory. With the aim of generalizing the theory to dense classes we study structurally sparse graph classes, which are defined as appropriate transductions of sparse graph classes. In this talk we quickly review the classical results from formal language and automata theory, which identify classes of bounded cliquewidth as MSO-transductions of trees. We then discuss FO-transductions of sparse graph classes and the connections with classical model theory, in particular with stability theory.

1. MSO-transductions of trees

Logical transductions offer a powerful tool to translate results from one class of structures to another. In this work we are particularly interested in structural graph theory, hence, we consider only classes of finite graphs (finite structures over a relation with one binary relation symbol E representing the edge relation). For our purposes it is sufficient to consider the following restricted form of transductions.

Definition 1.1 *A* simple transduction T with m parameters *is a formula* $\varphi(x, y, Z_1, \ldots, Z_m)$*, where* x, y *are two free first-order variables and* $Z_1, \ldots, Z_m$ *are free set variables. For a graph* G *and parameters* $P_1, \ldots, P_m \subseteq V(G)$*, we define* $\mathsf{T}(G, P_1, \ldots, P_m)$ *as the graph on vertex set* $V(G)$ *with edge set* $\{\{u, v\} : G \models \varphi(u, v, P_1, \ldots, P_m)$ *or* $G \models \varphi(v, u, P_1, \ldots, P_m), u, v \in V(G)\}$.

If φ is a formula of a logic $\mathcal{L}$, then we speak of an $\mathcal{L}$-transduction. In this work we will deal with first-order logic (FO) and monadic second-order logic (MSO).

In general, one can consider more general transductions that allow to replace a structure $\mathfrak{A}$ by the union of a fixed number of disjoint copies of $\mathfrak{A}$, augmented with appropriate relations between the copies, that define more general relations and restrictions of the universe. In model theory one commonly studies *interpretations* that usually come without set parameters. Copy operations can be simulated by interpretations in powers and by building quotients. We refer to the literature for more background on structures, logic, transductions and interpretations [6, 13].

Definition 1.2 *Let* $\mathscr{C}$ *and* $\mathscr{D}$ *be classes of graphs. We say that* $\mathscr{C}$ *is an* $\mathcal{L}$-transduction *of* $\mathscr{D}$ *if there exists an* $\mathcal{L}$*-transduction* T *with* m *parameters for some* $m \in \mathbb{N}$ *such that for every* $G \in \mathscr{C}$ *there exists* $H \in \mathscr{D}$ *and* $P_1, \ldots, P_m \subseteq V(H)$ *such that* G *is an induced subgraph of* $\mathsf{T}(H, P_1, \ldots, P_m)$*. We write* $\mathscr{C} \sqsubseteq_{\mathcal{L}} \mathscr{D}$ *if* $\mathscr{C}$ *is an* $\mathcal{L}$*-transduction transduction of* $\mathscr{D}$.

If graphs are replaced by their incidence graphs, MSO-formulas become more powerful, because they can quantify over sets of edges. For each graph G we write G_{in} for the incidence representation of G, and for a class $\mathscr{C}$ of graphs we write $\mathscr{C}_{\text{in}}$ for the class containing all incidence representations of graphs in $\mathscr{C}$. We write $\mathscr{C} \sqsubseteq^{\text{in}}_{\mathcal{L}} \mathscr{D}$ if $\mathscr{C}_{\text{in}} \sqsubseteq_{\mathcal{L}} \mathscr{D}_{\text{in}}$. For $n \in \mathbb{N}$ we denote the class of trees of height n by $\mathscr{T}_n$, the class of paths by $\mathscr{P}$, and the class of trees by $\mathscr{T}$.

Theorem 1.3 (see e.g. [3, 5, 6]) *Every class $\mathscr{C}$ of graphs in incidence representation can be encoded by MSO-transductions in one of the following classes.*

- *If $\mathscr{C}$ satisfies $\mathscr{C} \sqsubseteq^{\text{in}}_{\text{MSO}} \mathscr{T}_n$ for some $n \in \mathbb{N}$, then $\mathscr{C}$ is called a class of* bounded treedepth.
- *If $\mathscr{C}$ satisfies $\mathscr{C} \sqsubseteq^{\text{in}}_{\text{MSO}} \mathscr{P}$, then $\mathscr{C}$ is called a class of* bounded pathwidth.
- *If $\mathscr{C}$ satisfies $\mathscr{C} \sqsubseteq^{\text{in}}_{\text{MSO}} \mathscr{T}$, then $\mathscr{C}$ is called a class of* bounded treewidth.
- *Every finite graph can be encoded in a sufficiently large finite square grid (by a fixed transduction T).*

When considering the standard encoding we obtain the following classification.

- *If $\mathscr{C}$ satisfies $\mathscr{C} \sqsubseteq_{\text{MSO}} \mathscr{T}_n$ for some $n \in \mathbb{N}$, then $\mathscr{C}$ is called a class of* bounded shrubdepth.
- *If $\mathscr{C}$ satisfies $\mathscr{C} \sqsubseteq_{\text{MSO}} \mathscr{P}$, then $\mathscr{C}$ is called a class of* bounded linear cliquewidth.
- *If $\mathscr{C}$ satisfies $\mathscr{C} \sqsubseteq_{\text{MSO}} \mathscr{T}$, then $\mathscr{C}$ is called a class of* bounded cliquewidth.

We refer to the literature for background on the above defined graph theoretic concepts. For those familiar with transductions, the above definitions may be even more convenient than the definitions by tree models, tree decompositions and clique expressions.

Hence, from a logical point of view we can see the notions of shrubdepth, linear cliquewidth and cliquewidth as dense analogues of treedepth, pathwidth and treewidth. These analogies can also be seen from the graph theoretic point of view as follows. We call a class $\mathscr{C}$ of graphs *weakly sparse* if and only if there is $t \in \mathbb{N}$ such that every $G \in \mathscr{C}$ excludes the complete bipartite graph $K_{t,t}$ as a subgraph.

Theorem 1.4 *Let $\mathscr{C}$ be a weakly sparse graph class. Then*

- *$\mathscr{C}$ has bounded treedepth if and only if $\mathscr{C}$ has bounded shrubdepth [8].*
- *$\mathscr{C}$ has bounded pathwidth if and only if $\mathscr{C}$ has bounded linear cliquewidth [12].*
- *$\mathscr{C}$ has bounded treewidth if and only if $\mathscr{C}$ has bounded cliquewidth [12].*

The power of MSO in transductions on trees is in fact only needed to define the least common ancestor of two vertices. First-order logic can express only local properties and in particular in general cannot find the least common ancestor of two vertices. For paths and general trees we must change the representation to increase the power of FO accordingly. We write $\mathscr{T}^{\leq}$ for the class of all trees represented by tree orders, i.e., by the relation $u \leq v$ if u is an ancestor of v, and $\mathscr{P}^{\leq}$ for the class of all linear orders.

Theorem 1.5 *Let $\mathscr{C}$ be a class of graphs.*

- *$\mathscr{C}$ has bounded treedepth if and only if $\mathscr{C} \sqsubseteq^{\text{in}}_{\text{FO}} \mathscr{T}_n$ for some $n \in \mathbb{N}$ and it has bounded shrubdepth if and only if $\mathscr{C} \sqsubseteq_{\text{FO}} \mathscr{T}_n$ for some $n \in \mathbb{N}$ [9].*
- *$\mathscr{C}$ has bounded pathwidth if and only if $\mathscr{C} \sqsubseteq^{\text{in}}_{\text{FO}} \mathscr{P}^{\leq}$ and $\mathscr{C}$ has bounded linear cliquewidth if and only if $\mathscr{C} \sqsubseteq_{\text{FO}} \mathscr{P}^{\leq}$ [4].*
- *$\mathscr{C}$ has bounded treewidth if and only if $\mathscr{C} \sqsubseteq_{\text{FO}} \mathscr{T}^{\leq}$ and $\mathscr{C}$ has bounded cliquewidth if and only if $\mathscr{C} \sqsubseteq_{\text{FO}} \mathscr{T}^{\leq}$ [4].*

This naturally raises the question about the expressive power of FO on paths and trees, or in other words, on the role of order in the above classification. This question brings us to classical model theory, in particular, to a branch of model theory called *classification theory* or *stability theory*.

2. FO-transductions

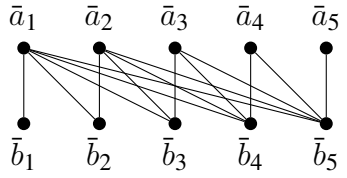
Stability theory aims to classify first-order theories by their difficulty. One of the key dividing lines between *tame* and *wild* theories is the ability of FO to define linear orders. Theories where no large orders can be defined are called *stable*. In our graph theoretic context, the concept can be defined as follows.

Definition 2.1 *A class $\mathscr{C}$ of graphs is called* stable *if for every first-order formula $\varphi(\bar{x}, \bar{y})$ there exists an integer k such that for every graph $G \in \mathscr{C}$ and for all tuples $\bar{a}_1, \ldots, \bar{a}_\ell \in V(G)^{|\bar{x}|}$ and $\bar{b}_1, \ldots, \bar{b}_\ell \in V(G)^{|\bar{y}|}$, if*

$$G \models \varphi(\bar{a}_i, \bar{b}_j) \quad \iff \quad i \leq j \tag{1}$$

for all $i, j \in [\ell]$, then $\ell \leq k$.

The combinatorial obstacle in this definition is called a *halfgraph* or *ladder* (of order ℓ). From a halfgraph we can define an order on tuples by $G \models \bar{a}_i\bar{b}_i < \bar{a}_j\bar{b}_j \Leftrightarrow G \models \varphi(\bar{a}_i, \bar{b}_j) \wedge \neg\varphi(\bar{a}_j, \bar{b}_i)$.



Still reasonably behaved theories are dependent theories, which in our context are defined as follows.

Definition 2.2 *A class $\mathscr{C}$ of graphs is called* dependent *(or* NIP*) if for every first-order formula $\varphi(\bar{x}, \bar{y})$ there exists an integer k such that for every graph $G \in \mathscr{C}$ and for all tuples $\bar{a}_i \in V(G)^{|\bar{x}|}$ ($i \in [\ell]$) and, $\bar{b}_I \in V(G)^{|y|}$ ($I \subseteq [\ell]$), if*

$$G \models \varphi(\bar{a}_i, \bar{b}_I) \quad \iff \quad i \in I \tag{2}$$

for all $i \in [\ell]$ and all $I \subseteq [\ell]$, then $\ell \leq k$.

A stronger notion of stability and of dependence arises when one allows to apply monadic lifts to the graphs in $\mathscr{C}$ before using the formula φ. These variants are called *monadic stability* and *monadic dependence*. The additional expressive power gained by the introduction of a monadic lift is so important that tuples of free variables can be replaced by single free variables in the above definitions [2]. This fits exactly to our notion of transductions and we have the following. We write $\mathscr{H}$ for the class of all halfgraphs and $\mathscr{G}$ for the class of all graphs.

Definition 2.3

- *A class $\mathscr{C}$ of graphs is called* monadically stable *if and only if $\mathscr{H} \not\sqsubseteq_{\mathrm{FO}} \mathscr{C}$.*
- *A class $\mathscr{C}$ of graphs is called* monadically dependent *if and only if $\mathscr{G} \not\sqsubseteq_{\mathrm{FO}} \mathscr{C}$.*

Prominent examples of monadically stable classes are nowhere dense graph classes and classes of bounded expansion [1, 18, 19]. We refer to [17] for extensive background on these graph classes. Classes of bounded cliquewidth are monadically dependent, see e.g. [11].

We recently obtained the following characterization of stable classes of bounded linear cliquewidth, explaining the role of order in these graph classes. We say that a bipartite graph H with parts A and B is semi-induced in a graph G, if H is a subgraph of G and the edges between A and B in G are induced, that is, if for $a \in A$ and $b \in B$ we have $\{a,b\} \in E(H) \Leftrightarrow \{a,b\} \in E(G)$. We write $\mathscr{P}^n$ for the class of pathwidth at most n and $\mathscr{T}^n$ for the class of treewidth at most n.

Theorem 2.4 ([15]) *Let $\mathscr{C}$ be a class with bounded linear cliquewidth. Then the following are equivalent.*

- *$\mathscr{C}$ is stable.*
- *$\mathscr{C}$ excludes some half-graph as semi-induced subgraph.*
- *$\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathscr{P}^n$ for some $n \in \mathbb{N}$, i.e., $\mathscr{C}$ is an FO-transduction of a class of bounded pathwidth.*

This leads naturally to the following conjecture.

Conjecture 1 *Let $\mathscr{C}$ be a class with bounded cliquewidth. Then the following are equivalent.*

- *$\mathscr{C}$ is stable.*
- *$\mathscr{C}$ excludes some half-graph as semi-induced subgraph.*
- *$\mathscr{C} \sqsubseteq_{\mathrm{FO}} \mathscr{T}^n$ for some $n \in \mathbb{N}$, i.e., $\mathscr{C}$ is an FO-transduction of a class of bounded treewidth.*

We now turn to nowhere dense and bounded expansion classes. First, the weakly sparse classes among the monadically dependent and stable classes are especially well behaved.

Theorem 2.5 ([15]) *Let $\mathscr{C}$ be a weakly sparse graph class. Then the following are equivalent.*

- *$\mathscr{C}$ is monadically dependent.*
- *$\mathscr{C}$ is monadically stable.*
- *$\mathscr{C}$ is nowhere dense.*

Our main goal is to obtain a structural characterization of first-order transductions of nowhere dense and bounded expansion classes. We say that these classes are *structurally nowhere dense* and have *structurally bounded expansion*. Both are monadically stable. In particular, we are interested in extending the model-checking results from these classes [7, 10] to their structurally bounded counterparts. For classes with structurally bounded expansion we have obtained a nice characterization in terms of covers (or colorings).

A *p-cover*, for $p \in \mathbb{N}$, of a graph G is a family $\mathcal{U}_G$ of subsets of $V(G)$ such that if every set of at most p elements of G is contained in some $U \in \mathcal{U}_G$. If $\mathscr{C}$ is a class of graphs, then a *p-cover* of $\mathscr{C}$ is a family $\mathcal{U} = (\mathcal{U}_G)_{G \in \mathscr{C}}$, where $\mathcal{U}_G$ is a p-cover of G. A cover $\mathcal{U}$ is *finite* if $\sup\{|\mathcal{U}_{\mathbf{A}}| \,:\, \mathcal{G} \in \mathscr{C}\}$ is finite. Let $\mathscr{C}[\mathcal{U}]$ denote the class of graphs $\{G[U] \,:\, G \in \mathscr{C}, U \in \mathcal{U}_G\}$. If $\mathscr{W}$ is a class of graphs, we say that a cover $\mathcal{U}$ is a $\mathscr{W}$-cover if $\mathscr{C}[\mathcal{U}] \subseteq \mathscr{W}$.

Theorem 2.6 ([16]) *A class $\mathscr{C}$ of graphs has bounded expansion if and only if*

- *for every $p \in \mathbb{N}$ there is a class $\mathscr{D}^p$ of bounded treedepth such that $\mathscr{C}$ admits a $\mathscr{D}^p$-p-cover.*
- *for every $p \in \mathbb{N}$ there is a class $\mathscr{P}^p$ of bounded pathwidth such that $\mathscr{C}$ admits a $\mathscr{P}^p$-p-cover.*
- *for every $p \in \mathbb{N}$ there is a class $\mathscr{T}^p$ of bounded treewidth such that $\mathscr{C}$ admits a $\mathscr{T}^n$-p-cover.*

With the above introduction the following will not come as a surprise.

Theorem 2.7 *A monadically stable class $\mathscr{C}$ of graphs has structurally bounded expansion if and only if*

- *for every $p \in \mathbb{N}$ there is a class $\mathscr{D}^p$ of bounded shrubdepth such that $\mathscr{C}$ admits a $\mathscr{D}^p$-p-cover [8].*
- *for every $p \in \mathbb{N}$ there is a class $\mathscr{P}^p$ of bounded linear cliquewidth such that $\mathscr{C}$ admits a $\mathscr{P}^p$-p-cover [15].*
- *Conjecture: for every $p \in \mathbb{N}$ there is a class $\mathscr{T}^p$ of bounded cliquewidth such that $\mathscr{C}$ admits a $\mathscr{T}^p$-p-cover.*

We studied classes with bounded cliquewidth covers in [14].

3. Conclusion

While classes of bounded treedepth, pathwidth and treewidth, as well as classes of bounded shrubdepth, linear cliquewidth and cliquewidth are very well understood as MSO-transductions of bounded depth trees, paths and general trees, much less is known about FO-transductions of sparse graph classes. The notions of FO-transductions fit very well into the framework of monadic stability and monadic dependence that is studied in classical stability theory. Nevertheless, these notions are usually studied in the context of infinite structures and we are only beginning to develop a structural and algorithmic theory of finite graphs that have these properties. I hope that with this short note I could wake the readers enthusiasm for the topic.

References

[1] H. ADLER, I. ADLER, Interpreting nowhere dense graph classes as a classical notion of model theory. *European Journal of Combinatorics* **36** (2014), 322–330.

[2] J. T. BALDWIN, S. SHELAH, Second-order quantifiers and the complexity of theories. *Notre Dame Journal of Formal Logic* **26** (1985) 3, 229–303.

[3] A. BLUMENSATH, B. COURCELLE, On the Monadic Second-Order Transduction Hierarchy. *Logical Methods in Computer Science* **6** (2010) 2.

[4] T. COLCOMBET, A combinatorial theorem for trees. In: *International Colloquium on Automata, Languages, and Programming*. Springer, 2007, 901–912.

[5] B. COURCELLE, The monadic second-order logic of graphs VII: Graphs as relational structures. *Theoretical Computer Science* **101** (1992) 1, 3–33.

[6] B. COURCELLE, J. ENGELFRIET, *Graph structure and monadic second-order logic: a language-theoretic approach*. 138, Cambridge University Press, 2012.

[7] Z. DVOŘÁK, D. KRÁĽ, R. THOMAS, Deciding First-Order Properties for Sparse Graphs. In: *51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*. 2010, 133–142.

[8] J. GAJARSKÝ, S. KREUTZER, J. NEŠETŘIL, P. OSSONA DE MENDEZ, M. PILIPCZUK, S. SIEBERTZ, S. TORUŃCZYK, First-order interpretations of bounded expansion classes. In: *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 126:1–126:14.

[9] R. GANIAN, P. HLINĚNÝ, J. NEŠETŘIL, J. OBDRŽÁLEK, P. OSSONA DE MENDEZ, R. RAMADURAI, When Trees Grow Low: Shrubs and Fast MSO_1. In: *International Symposium on Mathematical Foundations of Computer Science*. Lecture Notes in Computer Science 7464, Springer-Verlag, 2012, 419–430.

[10] M. GROHE, S. KREUTZER, S. SIEBERTZ, Deciding first-order properties of nowhere dense graphs. In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. ACM, 2014, 89–98.

[11] M. GROHE, G. TURÁN, Learnability and definability in trees and similar structures. *Theory of Computing Systems* **37** (2004) 1, 193–220.

[12] F. GURSKI, E. WANKE, *The Tree-Width of Clique-Width Bounded Graphs without* $K_{n,n}$, Lecture Notes in Computer Science 1928, Springer, 2000, 196–205.

[13] W. HODGES, H. WILFRID, et al., *Model theory*. 42, Cambridge University Press, 1993.

[14] O. KWON, M. PILIPCZUK, S. SIEBERTZ, On low rank-width colorings. In: *Graph-theoretic concepts in computer science*. Lecture Notes in Comput. Sci. 10520, Springer, 2017, 372–385.

[15] J. NEŠETŘIL, P. O. DE MENDEZ, R. RABINOVICH, S. SIEBERTZ, Classes of graphs with low complexity: the case of classes with bounded linear rankwidth. *submitted* (2019).

[16] J. NEŠETŘIL, P. OSSONA DE MENDEZ, Grad and Classes with Bounded Expansion I. Decompositions. *European Journal of Combinatorics* **29** (2008) 3, 760–776.

[17] J. NEŠETŘIL, P. OSSONA DE MENDEZ, *Sparsity (Graphs, Structures, and Algorithms)*. Algorithms and Combinatorics 28, Springer, 2012. 465 pages.

[18] M. PILIPCZUK, S. SIEBERTZ, S. TORUŃCZYK, On the number of types in sparse graphs. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, 2018, 799–808.

[19] K.-P. PODEWSKI, M. ZIEGLER, Stable graphs. *Fund. Math.* **100** (1978), 101–107.

THEORIE-AG 2019

S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 22–22.

Universität Bremen

Balancing Straight-Line Programs

Moses Ganardi[A] Artur Jeż[B] Markus Lohrey[A]

[A]University of Siegen, Germany
{ganardi,lohrey}@eti.uni-siegen.de

[B]University of Wrocław, Poland
aje@cs.uni.wroc.pl

A straight-line program (SLP for short) is a context-free string grammar that produces exactly one string. SLPs are widely used in the areas of data compression and algorithmics on compressed data. The size of an SLP (usually measured as the sum of the lengths of all right-hand sides) for a string s can be much smaller than the length of s. This happens in particular, if s contains many repetitions of a pattern. String compression using SLPs is also tightly related to dictionary based compression, in particular the well-known LZ77 and LZ78 compressors. Besides the size of an SLP, also its depth is important. The depth of an SLP is the height of its derivation tree. The running time or space consumption of many algorithms for SLPs depends on the depth of the input SLP. Trivially, the depth of an SLP is upper bounded by its size. Moreover, an SLP for a string of length n must have depth $\log(n)$. We prove that SLPs can be balanced in the following sense: Given an SLP of size m for a string of length n one can compute in linear time an equivalent SLP of size $O(m)$ and depth $O(\log n)$. This improves a previous result of Rytter (where the size of the SLP was $O(m \cdot \log n)$) and solves a long-standing open problem from the area of grammar-based compression. If time permits, we will also discuss analogous balancing results from the area of grammar-based tree compression.

A short version of this paper will appear in the Proceedings of FOCS 2019. A long version can be found at https://arxiv.org/abs/1902.03568.

[A]Markus Lohrey has been supported by the DFG research project LO 748/10-1.

[B]Artur Jeż was supported under National Science Centre, Poland project number 2017/26/E/ST6/00191.

THEORIE-AG 2019

S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 23–23.

Universität Bremen

Decidability and Complexity of $\mathcal{ALCOIF}$ with Transitive Closure

Jean Christoph Jung[(A)] Carsten Lutz[(A)] Thomas Zeume[(B)]

[(A)]Universität Bremen
{jeanjung,clu}@uni-bremen.de

[(B)]TU Dortmund
thomas.zeume@cs.tu-dortmund.de

Abstract

We study decidability and complexity of the description logic $\mathcal{ALCOIF}$ extended with transitive closure. In particular, we show decidability and NEXPTIME-completeness of finite and unrestricted satisfiability. Our results are interesting and relevant beyond the area of description logics, for two reasons. First, the logic we study is a notational variant of propositional dynamic logic (PDL) extended with converse, nominals, and deterministic programs, subject to the restriction that programs of the form α^* contain either no deterministic program or only a single atomic program (and possibly its converse). It is worth mentioning that the decidability of the logic without the latter restriction is still open, and that the μ-calculus with the same three extensions is undecidable. The second interesting perspective is provided by the fact that $\mathcal{ALCOIF}$ is an expressive fragment of C^2, two variable first-order logic with counting quantifiers, and in fact even of the guarded fragment GC^2 thereof. It is known that GC^2 easily becomes undecidable when (an unrestricted number of) relations can be declared to have special semantic properties such as being a linear order, a transitive relation, or an equivalence relation. In some cases, decidability can be attained when only a limited number of special relations is admitted. In the logic studied in this paper, it is possible to express that a role is transitive, an equivalence relation, a linear order, or a forest, respectively, possibly using auxiliary symbols. Thus, our results show that (finite) satisfiability of $\mathcal{ALCOIF}$ remains decidable when we admit that an unbounded number of relations is declared to have any of the mentioned semantic properties, in marked contrast to C^2 and GC^2. In fact, they raise the question for a maximal logic with this property.

This paper has been presented at DL 2019.

THEORIE-AG 2019
S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 24–27.
Universität Bremen

Tissue P Systems with Anti-Cells

Artiom Alhazov[(A)] Rudolf Freund[(B)] Sergiu Ivanov[(C)]

[(A)]Institute of Mathematics and Computer Science
Academiei 5, Chişinău, MD-2028, Moldova
artiom@math.md

[(B)]TU Wien, Institut für Logic and Computation
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

[(C)]IBISC, Université Évry, Université Paris-Saclay
23, boulevard de France, 91034 Évry, France
sergiu.ivanov@univ-evry.fr

Abstract

The concept of a matter object being annihilated when meeting its corresponding anti-matter object is taken over for tissue P systems with cell division rules and cell / anti-cell annihilation rules. As derivation modes we may take any of the maximally parallel derivation modes as well as any of the maximal set derivation modes (non-extendable (multi)set of rules, (multi)set with maximal number of rules, (multi)set of rules affecting the maximal number of objects).

1. Introduction

For an introduction to the field of P systems we refer to the monograph [2] and the handbook of membrane systems [3]; for actual news and results we refer to the P systems webpage [4] as well as to the Bulletin of the International Membrane Computing Society.

In [1], the concept of anti-matter was introduced: for any object a *(matter)*, its anti-object *(anti-matter)* a^- is considered together with the corresponding *annihilation rule* $aa^- \to \lambda$. This annihilation rule is assumed to be a special non-cooperative rule having priority over all other rules in the sense of weak priority i.e., other rules then also may be applied if objects cannot be bound by some annihilation rule any more.

Natural numbers can be represented by the corresponding number of cells with a specific label in a tissue P system. Hence, in this paper we take over the idea of anti-objects for cells, i.e., for every cell $\bigcirc_h$ we take the anti-cell $\bigcirc_{h^-}$ and the cell / anti-cell annihilation rule $\bigcirc_h \bigcirc_{h^-} \to \lambda$. In the simplest case, we only use cell division, i.e., rules of the form $\bigcirc_h \to \bigcirc_{h'} \bigcirc_{h''}$, possibly also allowing cell renaming rules of the form $\bigcirc_h \to \bigcirc_{h'}$ and cell deletion rules of the form $\bigcirc_h \to \lambda$.

Natural numbers can be represented by the corresponding number of cells with a specific label. In this setting, computational completeness results can be obtained for tissue P systems with cell division rules and cell / anti-cell annihilation rules only. As derivation modes we may take any of the maximally parallel derivation modes as well as any of the maximally parallel set derivation modes.

2. Tissue P Systems with Cell Division and Anti-Cells

Only the following types of rules for cells in the tissue P system are used:

cell division $\bigcirc_h \rightarrow \bigcirc_{h'} \bigcirc_{h''}$: the cell $\bigcirc_h$ is divided into two cells, possibly changing the label h of the parent cell $\bigcirc_h$ to two new labels h', h'' for the child cells $\bigcirc_{h'}$ and $\bigcirc_{h''}$

changing cell label $\bigcirc_h \rightarrow \bigcirc_{h'}$: the label h of cell $\bigcirc_h$ is changed to h'

cell deletion $\bigcirc_h \rightarrow \lambda$: the cell $\bigcirc_h$ is deleted

cell / anti-cell $\bigcirc_h \bigcirc_{h^-} \rightarrow \lambda$: the cell $\bigcirc_h$ and its anti-cell $\bigcirc_{h^-}$ annihilate each other

We may use any of the following derivation modes:

max take a non-extendable multiset of rules

max_{rules} take a non-extendable multiset of rules with the maximal number of rules possible

$max_{objects}$ take a non-extendable multiset of rules affecting the maximal number of objects

$setmax$ take a non-extendable set of rules

$setmax_{rules}$ take a non-extendable set of rules with the maximal number of rules possible

$setmax_{objects}$ take a non-extendable set of rules affecting the maximal number of objects

Formally, a *tissue P system with anti-cells* (a *tPAC* for short) is a construct $\Pi = (H, w_0, R)$ where H is the set of *cell labels* used in the rules specified in R, w_0 is the initial set of cells with labels from H, and R is the set of rules of the forms described above, with the labels of the cells taken from H. In any computation step of Π a multiset or set of rules is chosen from the set R in such a way that it is consistent with the derivation mode. We emphasize that we assume cell / anti-cell annihilation rules to have weak priority over all other rules, i.e., as long as cell / anti-cell annihilation rules may bind some cells, other rules are not allowed to yet be taken into the multiset or set of rules constructed to be applied.

A *configuration* of the system can be represented by the currently existing cells. Starting from a given *initial configuration* and applying evolution rules as described above in the given derivation mode, we get *transitions* among configurations; a sequence of transitions forms a *computation*. A computation is *halting* if it reaches a configuration where no rule can be applied any more. In the *generative case*, a halting computation has associated a result, in the form of the number of cells present in the system; their numbers represents a vector of natural numbers. In the *accepting case*, all (vectors of) non-negative integers are accepted whose input, given as the corresponding numbers of initial cells in addition to w_0, leads to a halting computation. The set of non-negative integers and the set of (Parikh) vectors of non-negative integers generated/accepted as results of halting computations in Π using the derivation mode γ are denoted by $N_{\gamma,\delta}(\Pi)$ and $Ps_{\gamma,\delta}(\Pi)$, respectively, with $\delta \in \{gen, acc\}$. The corresponding families of sets of non-negative integers and the sets of vectors of non-negative integers generated/accepted by tPACs are denoted by $N_{\gamma,\delta}(\text{tPAC})$ and $Ps_{\gamma,\delta}(\text{tPAC})$, respectively.

3. Results

As a first result, we observe that rules changing a cell label, i.e., $\bigcirc_h \rightarrow \bigcirc_{h'}$, and cell deletion rules, i.e., $\bigcirc_h \rightarrow \lambda$, are not needed and can be replaced by using only cell division and suitable cell / cell annihilation rules.

Lemma 3.1 *Rules changing cell label, i.e.,* $\bigcirc_h \to \bigcirc_{h'}$*, and cell deletion rules, i.e.,* $\bigcirc_h \to \lambda$*, can be simulated by cell division and cell / anti-cell annihilation rules.*

Proof. A rule changing the cell label, i.e., $\bigcirc_h \to \bigcirc_{h'}$, can be simulated by the rules $\bigcirc_h \to \bigcirc_{h'} \bigcirc_{h''}$, $\bigcirc_{h''} \to \bigcirc_g \bigcirc_{g^-}$, and $\bigcirc_g \bigcirc_{g^-} \to \lambda$, where h'', g, g^- are new labels.

A cell deletion rule, i.e., $\bigcirc_h \to \lambda$, can be simulated by the rules $\bigcirc_h \to \bigcirc_g \bigcirc_{g^-}$ and $\bigcirc_g \bigcirc_{g^-} \to \lambda$, where g, g^- are new labels. □

A tPAC only using cell division and cell / anti-cell annihilation rules is called a *tPAC in normal form*. As an immediate consequence of Lemma 3.1 we obtain the following normal form theorem:

Theorem 3.2 *For every tPAC* Π *we can construct a tPAC* Π' *in normal form such that* $Y_{\gamma,\delta}(\Pi) = Y_{\gamma,\delta}(\Pi')$ *for any* $Y \in \{N, Ps\}$ *and any* $\delta \in \{gen, acc\}$ *as well as any*

$$\gamma \in \{max, max_{rules}, max_{objects}, setmax, setmax_{rules}, setmax_{objects}\}.$$

We now show that tPACs, even in normal form, characterize the families NRE and $PsRE$, respectively. The main proof idea – as used very often in the area of P systems – is to simulate (the computations of) register machines, as carried out in a similar way in [1] for P systems with anti-matter. The bracket notation [NF]tPACs indicates that the result holds for tPACs as well as for tPACs in normal form:

Theorem 3.3 *For any* $Y \in \{N, Ps\}$ *and* $\delta \in \{gen, acc\}$, $Y_{\gamma,\delta}$*([NF]tPAC)* $= YRE$ *for any*

$$\gamma \in \{max, max_{rules}, max_{objects}, setmax, setmax_{rules}, setmax_{objects}\}.$$

Proof. Let $M = (m, B, l_0, l_h, P)$ be a register machine, where m is the number of registers, B is the set of instruction labels, l_0 is the initial label, l_h is the final label, and P is the set of instructions. We now construct a tPAC Π which simulates (the computations of) M:

- $\Pi = (H, w_0, R)$;
- $H = \{r, r^- \mid 1 \leq r \leq m\} \cup \{l, l' \mid l \in B\} \cup \{\#, \#^-\}$ is the set of labels for the cells; the label $r, 1 \leq r \leq m$, is for the copies of cell $\bigcirc_r$ representing the contents of register r; the labels r^- are for the corresponding anti-cells;
- in the generating case, only the cell $\bigcirc_{l_0}$ is present at the beginning; in the accepting case, suitable copies of cells for representing the input vector are to be added;
- R contains the rules described in the following.

The contents of register r is represented by the number of copies of cells $\bigcirc_r$, $1 \leq r \leq m$, and for each cell $\bigcirc_r$ we also consider the corresponding anti-cell $\bigcirc_{r^-}$.

The instructions of M are simulated by the following rules in R_1:

- $l_1 : (ADD(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Simulated by the rules $\bigcirc_{l_1} \to \bigcirc_r \bigcirc_{l_2}$ and $\bigcirc_{l_1} \to \bigcirc_r \bigcirc_{l_3}$.
- $l_1 : (SUB(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq r \leq m$.
  As rules common for the simulations of all SUB-instructions, we have
  - $\bigcirc_{r^-} \to \bigcirc_{\#^-}$, $1 \leq r \leq m$,
  - the annihilation rules $\bigcirc_r \bigcirc_{r^-} \to \lambda$, $1 \leq r \leq m$, and $\bigcirc_\# \bigcirc_{\#^-} \to \lambda$,

– as well as the trap rules $\bigcirc_{\#^-} \to \bigcirc_{\#}\bigcirc_{\#}$ and $\bigcirc_{\#} \to \bigcirc_{\#}\bigcirc_{\#}$; these last two rules lead the system into an infinite computation whenever a cell with one of the trap symbols # or $\#^-$ is left without being annihilated.

(a) The *zero test* for instruction l_1 is simulated by the rules
$$\bigcirc_{l_1} \to \bigcirc_{l_1'} \bigcirc_{r^-} \text{ and } \bigcirc_{l_1'} \to \bigcirc_{\#}\bigcirc_{l_3}.$$
The cell labeled by #, generated by the second rule $\bigcirc_{l_1'} \to \bigcirc_{\#}\bigcirc_{l_3}$ can only be eliminated if the anti-cell $\bigcirc_{r^-}$ generated by the first rule $\bigcirc_{l_1} \to \bigcirc_{l_1'} \bigcirc_{r^-}$ is not annihilated by $\bigcirc_r$, i.e., only if register r is empty, which allows us to apply the rule $\bigcirc_{r^-} \to \bigcirc_{\#^-}$ and then the annihilation rule $\bigcirc_{\#}\bigcirc_{\#^-} \to \lambda$.

(b) The *decrement case* for instruction l_1 is simulated by the rule $\bigcirc_{l_1} \to \bigcirc_{l_2}\bigcirc_{r^-}$.
The anti-cell $\bigcirc_{r^-}$ either correctly annihilates one copy of cell $\bigcirc_r$, thus decrementing the register r, or else traps an incorrect guess by forcing the anti-cell $\bigcirc_{r^-}$ to evolve to $\bigcirc_{\#^-}$ and then to $\bigcirc_{\#}\bigcirc_{\#}$ in the next two steps in case register r is empty.

We finally observe that these two remaining derivation steps for trapping the decrement case as well as the remaining derivation step for correctly completing the decrement case or the zero test case do not influence the correct simulation of another SUB-instruction, even on the same register r, as the involved symbols have evolved at least one step before they could influence the symbols being generated by the new simulation sequence.

- $l_h : HALT$. Simulated by $\bigcirc_{l_h} \to \lambda$.

When the computation in M halts, the cell $\bigcirc_{l_h}$ is removed, and no further rules can be applied provided the simulation has been carried out correctly, i.e., if no cells labeled by trap symbols # are present in this situation. The remaining cells in the system represent the result computed by M.

Using Lemma 3.1 we can even transform the tPAC constructed above in a tPAC in normal form. We finally observe that throughout the proof the rules used in the simulations of register machine instructions need only be used once in each derivation step, so the construction works for the set modes, too. □

Finally we mention that computational completeness can also be extended from the generating and accepting case to the computing case, i.e., and tPAC, even in normal form, can also compute any partial recursive function or relation.

References

[1] A. ALHAZOV, B. AMAN, R. FREUND, P Systems with Anti-Matter. In: M. GHEORGHE, G. ROZENBERG, A. SALOMAA, P. SOSÍK, C. ZANDRON (eds.), *Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers*. Lecture Notes in Computer Science 8961, Springer, 2014, 66–85.

[2] GH. PĂUN, *Membrane Computing: An Introduction*. Springer, 2002.

[3] GH. PĂUN, G. ROZENBERG, A. SALOMAA (eds.), *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

[4] The P Systems Website. `http://ppage.psystems.eu/`.

THEORIE-AG 2019
S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 28–31.
Universität Bremen

Accepting Networks of Evolutionary Processors with Resources Restricted Filters

Bianca Truthe

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
bianca.truthe@informatik.uni-giessen.de

Abstract

In this paper, we continue the research on accepting networks of evolutionary processors where the filters belong to several special families of regular languages. These subregular families are defined by restricting the resources needed for generating or accepting them (the number of states of the minimal deterministic finite automaton accepting a language of the family as well as the number of non-terminal symbols or the number of production rules of a right-linear grammar generating such a language). We insert the newly defined language families into the hierachy of language families obtained by using as filters languages of other subregular families (such as ordered, non-counting, power-separating, circular, suffix-closed regular, union-free, definite, combinational, finite, monoidal, nilpotent, or commutative languages).

1. Introduction

Networks of language processors have been introduced in [2] by E. CSUHAJ-VARJÚ and A. SALOMAA. Such a network can be considered as a graph where the nodes represent processors which apply production rules to the words they contain. In a derivation step (an evolutionary step), any node derives from its language all possible words as its new language. In a communication step, any node sends those words to other nodes which satisfy an output condition given as a regular language (called the output filter) and any node adopts words sent by the other nodes if the words satisfy an input condition also given by a regular language (called the input filter).

Inspired by biological processes, in [1] a special type of networks of language processors was introduced. The nodes of such networks are called evolutionary processors because the allowed productions model the point mutation known from biology. The productions of a node allow that one letter is substituted by another letter, letters are inserted, or letters are deleted; the nodes are then called substitution nodes, insertion nodes, or deletion nodes, respectively.

Networks of evolutionary processors can be defined as language generating devices (the processors start working with finite sets of axioms and all words which are in a designated processor at some time form the generated language) and accepting ones (input words are accepted if there is a computation which leads to a word in a designated processor).

This is a short version of a paper presented at NCMA 2019 in Valencia (Spain) [4].

In [3], the computational power of accepting networks of evolutionary processors was investigated for cases that all filters belong to a certain subfamily of the set of all regular languages.

In [4], the research on accepting networks of evolutionary processors was continued where the filters are restricted by bounded resources, namely the number of non-terminal symbols or the number of production rules which are necessary for generating the languages or the number of states which are necessary for accepting the languages by deterministic finite automata. The language classes obtained by these restrictions are compared to those obtained in [3].

2. Preliminaries

Here, we explain accepting networks of evolutionary processors (ANEPs) and present the language families which are considered for the filters. Let V be an alphabet. By V^* we denote the set of all words over the alphabet V (including the empty word λ).

Intuitively, an accepting network over an alphabet V with n evolutionary processors is a graph consisting of n nodes (also called processors) and a set of directed edges between nodes. One of the nodes is the input node, one is the output node of the network. Any processor N_i with $1 \leq i \leq n$ consists of a set M_i of evolutionary rules, an input filter I_i, and an output filter O_i. We say that N_i is

- a substitution node if $M_i \subseteq \{a \to b \mid a, b \in V\}$ (by any rule, a letter is substituted by another one),
- a deletion node if $M_i \subseteq \{a \to \lambda \mid a \in V\}$ (by any rule, a letter is deleted), or
- an insertion node if $M_i \subseteq \{\lambda \to b \mid b \in V\}$ (by any rule, a letter is inserted).

Every node has rules from one type only. The input filter I_i and the output filter O_i control the words which are allowed to enter and to leave the node, respectively. With any node N_i and any time moment $t \geq 0$, we associate a set $C_t(i)$ of words (the words contained in the node at time t). Initially, the input node N_{n_i} contains the one word (the input word); the other nodes have no words. In an evolutionary step, we derive from $C_t(i)$ all words by applying rules from the set M_i. In a communication step, any processor N_i sends out all words from the set $C_t(i) \cap O_i$ (which pass the output filter) to all processors to which a directed edge exists (only the words from $C_t(i) \setminus O_i$ remain in the set associated with N_i) and, moreover, it receives from any processor N_k such that there is an edge from N_k to N_i all words sent by N_k and passing the input filter I_i of N_i, i. e., the processor N_i gets in addition all words of $C_t(k) \cap O_k \cap I_i$. We start with an evolutionary step and then communication steps and evolutionary steps are alternately performed. If, at some moment t with $t \geq 0$, the output node contains a word, then the input word is accepted. The accepted language is the set of all input words which are accepted by the network.

For a family X, we denote the family of languages accepted by networks of evolutionary processors where all filters are of type X by $\mathcal{A}(X)$.

In this paper, we consider filters which are defined by bounding the resources which are necessary for accepting or generating these languages. By RL_n^V, RL_n^P, and REG_n^Z, we denote the family of all languages which are generated by a right-linear grammar with at most n non-terminal symbols or production rules or accepted by a deterministic finite automaton with at most n states, respectively.

In [3], the following restrictions for regular languages are considered. In order to relate our results of this paper to the results there, we explain here those special regular languages. Let L be a language and V the minimal alphabet of L. We say that the language L, with respect to the alphabet V, is

- *monoidal* if $L = V^*$,
- *combinational* if it has the form $L = V^*A$ for some subset $A \subseteq V$,
- *definite* if it can be represented in the form $L = A \cup V^*B$ where A and B are finite subsets of V^*,
- *nilpotent* if L is finite or $V^* \setminus L$ is finite,
- *commutative* if $L = \{\, a_{i_1} \dots a_{i_n} \mid a_1 \dots a_n \in L,\ n \geq 1,\ \{i_1, i_2, \dots, i_n\} = \{1, 2, \dots, n\} \,\}$,
- *circular* if $L = \{\, vu \mid uv \in L,\ u, v \in V^* \,\}$,
- *suffix-closed* if the relation $xy \in L$ for some words $x, y \in V^*$ implies that also the suffix y belongs to L or equivalently, $L = \{\, v \mid uv \in L,\ u, v \in V^* \,\}$,
- *non-counting* (or star-free) if there is an integer $k \geq 1$ such that, for any $x, y, z \in V^*$, the relation $xy^k z \in L$ holds if and only if also $xy^{k+1}z \in L$ holds,
- *power-separating* if for any word $x \in V^*$ there is a natural number $m \geq 1$ such that either the equality $J_x^m \cap L = \emptyset$ or the inclusion $J_x^m \subseteq L$ holds where $J_x^m = \{x^n \mid n \geq m\}$,
- *ordered* if L is accepted by some finite automaton $\mathcal{A} = (Z, V, \delta, z_0, F)$ where $(Z, \preceq)$ is a totally ordered set and, for any $a \in V$, $z \preceq z'$ implies $\delta(z, a) \preceq \delta(z', a)$,
- *union-free* if L can be described by a regular expression which is only built by product and star.

Among the commutative, circular, suffix-closed, non-counting, and power-separating languages, we consider only those which are also regular.

By *FIN*, *MON*, *COMB*, *DEF*, *NIL*, *COMM*, *CIRC*, *SUF*, *NC*, *PS*, *ORD*, and *UF*, we denote the families of all finite, monoidal, combinational, definite, nilpotent, regular commutative, regular circular, regular suffix-closed, regular non-counting, regular power-separating, ordered, and union-free languages, respectively. Furthermore, *REG* and *RE* denote the families of all regular and all recursively enumerable languages, respectively.

The following theorem is known (see, e. g., [3]).

Theorem 2.1 *We have* $\mathcal{A}(REG) = RE$.

3. Summary of the Results

Accepting networks of evolutionary processors are investigated where the filters belong to subregular language families which are defined by restricting the resources needed for generating or accepting them (the number of states of the minimal deterministic finite automaton accepting a language of the family, the number of non-terminal symbols, or the number of production rules of a right-linear grammar generating such a language). These language families are inserted into the hierachy of language families obtained by using languages of other subregular families as filters (such as ordered, non-counting, power-separating, circular, suffix-closed regular, unionfree, definite, combinational, finite, monoidal, nilpotent, or commutative languages) which was published in [3]. The hierarchy with the new results is shown in Figure 1.

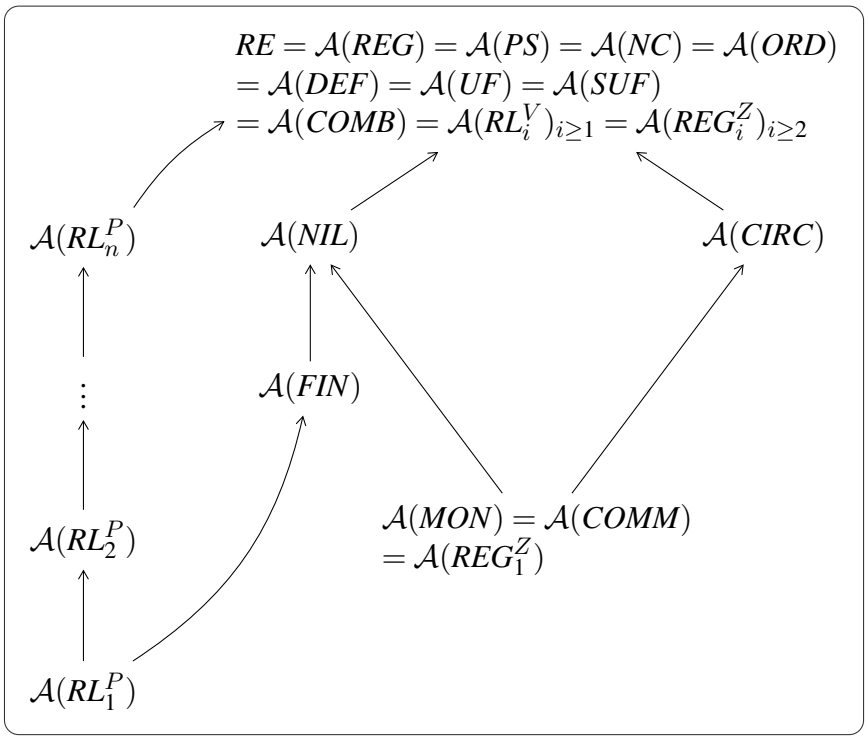


Figure 1: Hierarchy of language families by ANEPs with filters from subregular families. An arrow from a language family X to a language family Y stands for the proper inclusion $X \subset Y$. If two families X and Y are not connected by a directed path, then the families are incomparable.

References

[1] J. CASTELLANOS, C. MARTÍN-VIDE, V. MITRANA, J. M. SEMPERE, Solving NP-Complete Problems with Networks of Evolutionary Processors. In: *IWANN '01: Proceedings of the 6th International Work-Conference on Artificial and Natural Neural Networks*. LNCS 2084, Springer-Verlag Berlin, 2001, 621–628.

[2] E. CSUHAJ-VARJÚ, A. SALOMAA, Networks of Parallel Language Processors. In: *New Trends in Formal Languages – Control, Cooperation, and Combinatorics*. LNCS 1218, Springer-Verlag Berlin, 1997, 299–318.

[3] F. MANEA, B. TRUTHE, Accepting networks of evolutionary processors with subregular filters. *Theory of Computing Systems* **55** (2014) 1, 84–109.

[4] B. TRUTHE, Accepting Networks of Evolutionary Processors with Resources Restricted Filters. In: R. FREUND, M. HOLZER, J. M. SEMPERE (eds.), *11th Workshop on Non-Classical Models of Automata and Applications (NCMA), Valencia, Spain, July 2–3, 2019, Proceedings*. books@ocg.at 336, Österreichische Computer Gesellschaft, Austria, 2019, 187–202.

THEORIE-AG 2019
S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 32–35.
Universität Bremen

How are Eulerian trails connected to formal languages?

Meenakshi Paramasivan

Institut für Informatik, Universität Leipzig, D-04009 Leipzig, Germany
meena_maths@yahoo.com

Abstract

Some detailed proofs of few theorems in [1] and some additional results are given in [2].

1. Introduction and Definitions

We consider connected graphs. If we draw all the vertices of a graph $G = (V, E, \psi)$ on a horizontal line, then we associate different integers to the vertices by a function called *pseudo-linear drawing*, which is defined as follows:

Definition 1.1 *A* pseudo-linear drawing *(PLD) is an injective function* $\phi : V \to \mathbb{Z}$.

Definition 1.2 PLD$_\phi(G)$ *is a graph with vertex set* $\phi(V)$ *such that PLD*$_\phi(G) \cong G$.

Definition 1.3 *If* ϕ *is a PLD, then* u *is to the* left *of* v *in the drawing PLD*$_\phi(G)$ *if and only if* $\phi(u) < \phi(v)$. *Similarly* u *is to the* right *of* v *in the drawing PLD*$_\phi(G)$ *if and only if* $\phi(v) < \phi(u)$.

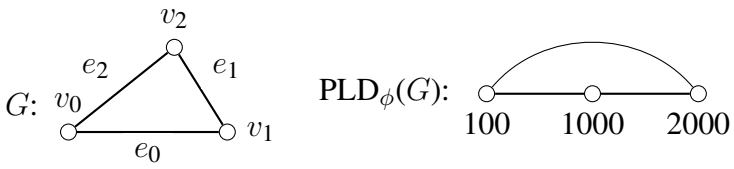


Figure 1: Graph G and PLD$_\phi(G)$ in Example 1.4

Example 1.4 *Let* G *be graph as in Fig. 1 and let* ϕ *be a PLD defined as* $\phi(v_0) = 100$, $\phi(v_1) = 1000$ *and* $\phi(v_2) = 2000$ *then PLD*$_\phi(G)$ *is as in Fig. 1.*

The word representation of the Eulerian trails is as follows:

Definition 1.5 *A connected graph $G = (V, E, \psi)$, with an Eulerian trail $W = v_0 e_0 \dots e_{k-1} v_k$ and any PLD $\phi : V \to \mathbb{Z}$, defines a* word $w = word(G, \phi, W) \in \Sigma^*$, $\Sigma = \{\rightarrow, \leftarrow, |\}$ *associated to G, ϕ and W as follows:*

$$word(G,\phi,W) = \begin{cases} \varepsilon & \text{if } W = v_0 \\ w'w'', w'' = \rightarrow |^s & \text{if } s = \phi(v_k) - \phi(v_{k-1}) \geq 0 \wedge W \neq v_0 \\ w'w'', w'' = \leftarrow |^s & \text{if } s = \phi(v_{k-1}) - \phi(v_k) > 0 \wedge W \neq v_0 \end{cases}$$

Here $w' = word(G', \phi', W')$, where $W' = v_0 e_0 \dots e_{k-2} v_{k-1}$, $G' \subsetneq G$,

$$G' = \begin{cases} G - e_{k-1} & \text{if } d_G(v_k) > 1 \\ G - v_k & \text{if } d_G(v_k) = 1 \end{cases}$$

with $V(G') = V' = V$ if $G' = G - e_{k-1}$ and $V' = V - v_k$ if $G' = G - v_k$, $E(G') = E' \subsetneq E$, ψ' is the restriction of ψ to E' and ϕ' the restriction of ϕ to V'.

Note 1: In the above definition the trail W satisfies $W = W' \cdot_{v_{k-1}} W''$ where $\ell(W') = k - 1$, and $W'' = v_{k-1} e_{k-1} v_k$ where $\ell(W'') = 1$ to satisfy $\ell(W) = k$.

Note 2: $W'' = v_{k-1} e_{k-1} v_k$ is an Eulerian trail of the graph $G'' = (V'', E'', \psi'')$, where $V'' = \{v_k, v_{k-1}\}$, $E'' = \{e_{k-1}\}$, $\psi''(e_{k-1}) = v_{k-1} v_k$. Then G'' satisfies the condition that $G = G' \cup G''$ with $v_{k-1} \in V' \cap V''$. Here G' and G'' are edge-disjoint but not vertex-disjoint. Also G'' has a PLD ϕ'', the restriction of ϕ to V'', that is $\phi'' = \phi|_{V''}$. So we have $word(G'', \phi'', W'') = w''$. This gives the following remark.

Remark 1.6 $word(G, \phi, W) = word(G', \phi', W') \cdot word(G'', \phi'', W'')$.

Hence, G, ϕ, and an Eulerian trail W in G specify a word $word(G, \phi, W) = w$ over the alphabet $\Sigma = \{\rightarrow, \leftarrow, |\}$, called *Eulerian trace*. This gives the following formal definition of the *set of all Eulerian traces*.

Definition 1.7 $ET = \{w \in \{\rightarrow, \leftarrow, |\}^* \mid \exists G = (V, E, \psi)$ *with an Eulerian trail W and $\exists$ a PLD $\phi : V \to \mathbb{Z}$ such that $w = word(G, \phi, W)$*$\}$.

2. Results

Lemma 2.1 $ET \subseteq \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$, *where* $\Sigma = \{\rightarrow, \leftarrow, |\}$.

Lemma 2.2 *Let W_1 and W_2 be two Eulerian trails of graphs $G_1 = (V_1, E_1, \psi_1)$ and $G_2 = (V_2, E_2, \psi_2)$ respectively. Then $W_1 \cdot_{v_k} W_2$ is also an Eulerian trail of graph $G_1 \cup G_2$ with $v_k \in V_1 \cap V_2$ and v_k being the terminus of W_1 and origin of W_2.*

Remark 2.3 Lemma 2.2 is true only for v_k being the terminus of W_1 and the origin of W_2 and it is not true for any $v_k \in V_1 \cap V_2$. For instance, let $W_1 = x_0 e_0 x_1 e_4 x_2 e_1 x_0 e_2 x_3$ and $W_2 = y_0 f_0 x_0 f_1 y_2$ (see Fig. 2). Here, x_0 is the vertex common in the vertex set of both W_1 and W_2. But it is not both: terminus of W_1 and origin of W_2, so $W_1 \cdot_{x_0} W_2$ is not an Eulerian trail (see Fig. 3).

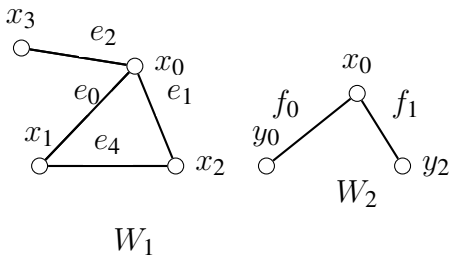


Figure 2: Eulerian trails W_1 and W_2

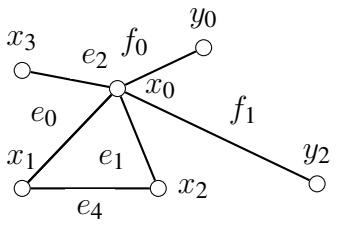


Figure 3: $W_1 \cdot_{x_0} W_2$

To prove the other inclusion of Lemma 2.1, let us define the homomorphism $h : \Sigma^* \to \{\rightarrow, \leftarrow\}^*$ such that $h(\rightarrow) = \rightarrow$, $h(\leftarrow) = \leftarrow$ and $h(|) = \varepsilon$. Let us consider $w \in \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$. We have to prove that $w \in ET$. For this let us restate $\Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\}) \subseteq ET$ in more detail as in the following lemma.

Lemma 2.4 *Let $w \in \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$. Let $k \in \mathbb{N}_0$. If $|h(w)| = k$, then w corresponds to an Eulerian trail W with $\ell(W) = \varepsilon(W) = k$ of a graph $G = (V, E, \psi)$ with $\nu(G) = \nu(W) = n$ where $k + 1 \geq n \geq 1$ and $\varepsilon(G) = k$ that has a PLD $\phi : V \to \mathbb{Z}$ such that $word(G, \phi, W) = w$.*

An illustration of Lemma 2.4 is in [2]

Theorem 2.5 $ET = \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$.

Proof. The proof follows from Lemmas 2.1 and 2.4. □

3. Standard PLD

In Section 1 we saw that given any connected graph $G = (V, E, \psi)$ with an Eulerian trail W and any PLD $\phi : V \to \mathbb{Z}$, we find a word $w \in ET$. Conversely, given any word $w \in ET$ we find the connected graph $G = (V, E, \psi)$ with an Eulerian trail W that has a PLD $\phi : V \to \mathbb{Z}$ such that $word(G, \phi, W) = w$.

Definition 3.1 *Let $\phi : V \to \mathbb{Z}$ be a PLD of the graph $G = (V, E, \psi)$ with an Eulerian trail W with $\ell(W) = \varepsilon(G) = k$ and $\nu(W) = \nu(G) = n$ where $k + 1 \geq n \geq 1$, $k \in \mathbb{N}_0$. Let $\phi(v_i) = z_i$ where $v_i \in V$ and $0 \leq i \leq k$. The* translation of ϕ, *$\phi_T : V \to \mathbb{Z}$ is defined by $\phi_T(v_i) = z_i + C$ where $C \in \mathbb{Z}$.*

Definition 3.2 *Let $\mathcal{P}$ be the set of all PLDs. Let ϕ_1, $\phi_2 \in \mathcal{P}$. We say that ϕ_1, ϕ_2 are equivalent, $\phi_1 \backsimeq \phi_2$ if and only if $\phi_1(v) - \phi_2(v) = d$ for some $d \in \mathbb{Z}$ and for all $v \in V$.*

As the name chosen in the previous definition suggests, we can prove that the relation $\backsimeq$ on $\mathcal{P}$ is an equivalence relation:

Lemma 3.3 *$\backsimeq \subseteq \mathcal{P} \times \mathcal{P}$ is an equivalence relation.*

Since $\backsimeq$ on the set $\mathcal{P}$ is an equivalence relation. For each $\phi_1 \in \mathcal{P}$ we can define the equivalence class of ϕ_1, denoted by $[\phi_1]$, to be the set

$$[\phi_1] = \{\phi_2 \in \mathcal{P} \mid \phi_2 \backsimeq \phi_1\}.$$

Definition 3.4 *A PLD ϕ_w of G_w is said to be a* standard PLD*, if and only if the start vertex v_0 of the Eulerian trail W of G_w satisfies the condition that $\phi_w(v_0) = 0$.*

4. Eulerian Traces

This section gives certain subsets of ET that describe few properties of the Eulerian traces.

Lemma 4.1 $ET \in \mathcal{REG}$.

Proof. By Theorem 2.5, $ET = \Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$. Since ET is expressed by a regular expression $\Sigma^* \setminus ((\{|\}\Sigma^*) \cup \{\leftarrow\})$, $\Sigma = \{\rightarrow, \leftarrow, |\}$, $ET \in \mathcal{REG}$. □

Definition 4.2 $ET^\circ = \{w \in \Sigma^* : w \in ET \wedge G_w$ *is Eulerian*$\}$, $\Sigma = \{\rightarrow, \leftarrow, |\}$.

Theorem 4.3 *ET° is a deterministic blind one-counter language.*

Definition 4.4 $ET_{\textit{loop-free}} = \{w \in \Sigma^* : w \in ET \setminus \{\varepsilon\} \wedge G_w$ *has no loops*$\}$.

Lemma 4.5 $\forall w \in ET \setminus \{\varepsilon\}$ $[w \in ((\rightarrow + \leftarrow)|^+)^+ \iff G_w$ *has no loops*$]$.

Theorem 4.6 $ET_{\textit{loop-free}} \in \mathcal{REG}$

References

[1] H. FERNAU, M. PARAMASIVAN, Formal Language Questions for Eulerian Trails. In: T. NEARY, M. COOK (eds.), *Machines, Computations and Universality, MCU*. Electronic Proceedings in Theoretical Computer Science EPTCS 128, Open Publishing Association, 2013, 25–26.

[2] M. PARAMASIVAN, *Operations on Graphs, Arrays and Automata*. Dissertation, Universität Trier, 2017.

THEORIE-AG 2019
S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 36–37.
Universität Bremen

Erweiterungen zu kleinen synchronisierenden Wörtern

Henning Fernau[(A)]

[(A)]Universität Trier, Abteilung Informatikwissenschaften, CIRT, 54296 Trier
{fernau}@uni-trier.de

Zusammenfassung

Erweiterungsprobleme wurden bislang vornehmlich für kombinatorische Probleme auf Graphen betrachtet. Hier wird diese Betrachtungsweise an synchronisierenden Wörtern erprobt. Im Gegensatz zu bisherigen Szenarien ergibt sich ein reichhaltiges komplexitätstheoretisches Bild in Abhängigkeit von der Wahl der Halbordnung auf der Menge aller Wörter.

1. Einleitung

Erweiterungsprobleme sind als fast schon spielerische Abart schwerer kombinatorischer Probleme bekannt; es sei nur an Lateinische Quadrate oder Spiele wie Sudoku erinnert. Immer geht es darum, eine vorgegebene Teillösung zu einer gültigen Lösung zu erweitern.

Im Bereich klassischer (graphentheoretischer) kombinatorischer Probleme kommt folgende Betrachtung als Motivation hinzu: Sowohl zum Auffinden von Lösungen als auch zum Auflisten aller (minimaler oder maximaler) Lösungen kommen oft Algorithmen zum Einsatz, die bestehende Teillösungen zu erweitern versuchen. Dabei ist es interessant zu wissen, ob eine Teillösung überhaupt zu einer interessierenden Lösung erweitert werden kann oder nicht.

Wir möchten ein konkretes Problem ansprechen und damit die Vorgehensweise erklären. Verfahren zur Lösung des Knotenüberdeckungsproblems arbeiten häufig so, dass für einen gewählten Knoten entschieden wird (durch Fallunterscheidung, d.h. also durch Verzweigen), ob er in die Lösung kommt oder nicht (im letzteren Falle müssen dann alle seine Nachbarn in die Lösungsmenge aufgenommen werden). Bevor nun rekursiv das Verfahren fortgesetzt wird, wäre es gut zu wissen, ob es überhaupt eine inklusionsminimale Überdeckung gibt, die alle bisher in die Teillösung aufgenommenen Knoten enthält. Leider (und möglicherweise ist das etwas erstaunlich) ist diese Fragestellung NP-schwer, selbst bei sehr eingeschränkten Graphklassen.

Abstrakter betrachtet liegt eine Halbordnung auf der Menge der Teillösungen vor (konkret die Inklusionshalbordnung auf der Menge der Knotenmengen beim Knotenüberdeckungsproblem), und es stellt sich die Frage, ob es eine (in dieser Halbordnung) größere Teillösung gibt, die eine gültige Lösung darstellt, zu der es aber keine (in dieser Halbordnung) echt kleinere gültige Lösung gibt. Diese Sichtweise überträgt sich ganz natürlich auch beispielsweise auf kombinatorische Probleme, bei denen Wörter (und nicht Mengen) Teillösungen darstellen. Im Gegensatz zu Mengen gibt es nun aber eine ganze Reihe natürlicher Halbordnungsbegriffe auf Wörtern. Diese Szenarien haben wir in dieser Arbeit im Kontext mit synchronisierenden Wörtern diskutiert.

[(A)]Eine Langfassung erscheint im Sonderband über synchronisierende Wörter bei JALC [2].

2. Erweiterungsprobleme bei synchronisierenden Wörtern

Ein *deterministischer endlicher Halbautomat* (DEH) wird beschrieben durch ein Tripel $A = (S, \Sigma, \delta)$, wobei S, Σ Alphabete sind und $\delta : S \times \Sigma \to S$ eine Abbildung ist, die in natürlicher Weise auch als $\delta : 2^S \times \Sigma^* \to 2^S$ aufgefasst werden kann. Ein Wort $w \in \Sigma^*$ heißt *synchronisierend*, falls $|\delta(S, w)| = 1$. Bekanntermaßen [3, 1] ist die Frage, of ein DEH ein synchronisierendes Wort der Länge höchstens k besitzt, NP-schwer. Ist nun $\prec$ eine Halbordnung auf Σ^*, so können wir das SW-Erweiterungsproblem bzgl. $\prec$, kurz EXT DFA-SW-$\prec$, folgendermaßen definieren: Gegeben ist ein DEH $A = (S, \Sigma, \delta)$ sowie ein Wort $u \in \Sigma^*$, gibt es $w \in \Sigma^*$, $u \prec w$, sodass w $\prec$-minimal ist in der Menge $L_{sync}(A)$ der synchronisierenden Wörter von A?

Wir betrachten diese Fragestellung für $\prec \in \mathcal{PO} := \{\sqsubseteq, \sqsupseteq, |, \mathit{sub}, \leq_{lex}, \leq_{ll}\}$, wobei

- $\sqsubseteq$ bzw. $\sqsupseteq$ die Präfix- bzw. Suffixrelation,
- $|$ die Teilfolgenrelation,
- *sub* die Teilwort- (oder Faktor-) Relation sowie
- $\leq_{lex}$ und $\leq_{ll}$ die eine vorgegebene Ordnung auf dem Alphabet fortsetzende lexikographische bzw. längenlexikographische Ordnung bezeichnet.

Wir können unsere Ergebnisse wie folgt zusammenfassen:

- EXT DFA-SW-$\sqsubseteq$ bzw. EXT DFA-SW-$\sqsupseteq$ sind in Polynomzeit entscheidbar.
- EXT DFA-SW-$|$ ist NP-schwer.
- Der Status von EXT DFA-SW-*sub* ist ungeklärt.
- EXT DFA-SW-$\leq_{lex}$ ist in Polynomzeit entscheidbar.
- EXT DFA-SW-$\leq_{ll}$ ist co-NP-schwer.

Ähnliche Ergebnisse kann man auch für verschiedene Einschränkungen endlicher Automaten festhalten. Einzelheiten hierzu finden sich in einer Arbeit, die gemeinsam mit Stefan Hoffmann entstand [2].

Literatur

[1] D. EPPSTEIN, Reset Sequences for Monotonic Automata. *SIAM Journal on Computing* **19** (1990) 3, 500–510.

[2] H. FERNAU, S. HOFFMANN, Extensions to minimal synchronizing words. *Journal of Automata, Languages and Combinatorics* **24** (2019), 287–307.

[3] I. K. RYSTSOV, On minimizing the length of synchronizing words for finite automata. In: *Theory of Designing of Computing Systems*. Institute of Cybernetics of Ukrainian Acad. Sci., 1980, 75–82. In russischer Sprache.

THEORIE-AG 2019

S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 38–41.

Universität Bremen

Computational Complexity of Synchronization under Regular Constraints

Henning Fernau[(A)] Vladimir V. Gusev[(B)] Stefan Hoffmann[(A)]
Markus Holzer[(C)] Mikhail V. Volkov[(D)] Petra Wolf[(A)]

[(A)]Universität Trier
{fernau, hoffmanns, wolfp}@uni-trier.de

[(B)]University of Liverpool
vladimir.gusev@liverpool.ac.uk

[(C)]Universität Gießen
holzer@informatik.uni-giessen.de

[(D)]Ural Federal University
m.v.volkov@urfu.ru

Abstract

Many variations of synchronization of finite automata have been studied in the previous decades. Here, we suggest studying the question if synchronizing words exist that belong to some fixed constraint language, given by some partial finite automaton called constraint automaton. We show that this synchronization problem becomes PSPACE-complete even for some constraint automata with two states and a ternary alphabet. In addition, we characterize constraint automata with arbitrarily many states for which the constrained synchronization problem is polynomial-time solvable. We classify the complexity of the constrained synchronization problem for constraint automata with two states and two or three letters completely and lift those results to larger classes of finite automata.

1. Introduction

Synchronization is an important concept for many applied areas: parallel and distributed programming, system and protocol testing, information coding, robotics, etc. At least some aspects of synchronization are captured by the notion of a synchronizing automaton; for instance, synchronizing automata adequately model situations in which one has to direct a certain system to a particular state without a priori knowledge of its current state. We only refer to some survey papers [8, 10] that report on some of these applications. An automaton is called synchronizing if there exists a word that brings it to a known state independently of the starting state. This concept is quite natural and has been investigated intensively in the last six decades. It is related

The second author is supported by the Leverhulme Trust. The last two authors are supported by the DFG-funded project FE560/9-1.

to the arguably most famous open combinatorial question in automata theory, formulated by Černý in [1]. The Černý conjecture states that every n-state synchronizing automaton can be synchronized by a word of length smaller or equal $(n-1)^2$. Although this bound was proven for several classes of finite-state automata, the general case is still widely open. The currently best upper bound on this length is cubic, and only very little progress has been made, basically improving on the multiplicative constant factor in front of the cubic term, see [9].

Due to the importance of this notion of synchronizing words, quite a large number of generalizations and modifications have been considered in the literature. We only mention four of these in the following. Instead of synchronizing the whole set of states, one could be interested in synchronizing only a subset of states. This and related questions were first considered by Rystsov in [7]. Instead of considering deterministic finite automata (DFAs), one could alternatively study the notion of synchronizability for nondeterministic finite automata [3, 6]. The notion of synchronizability naturally transfers to partially defined transition functions where a synchronizing automata avoiding undefined transitions is called *carefully synchronizing*, see [5, 6]. Recall that the question of synchronizability (without length bounds) is solvable in polynomial time for complete DFAs [10]. However, in all of the mentioned generalizations, this synchronizability question becomes even PSPACE-complete. This tendency can also be observed in the generalization that we introduce in this paper, which we call *regular constraints*. These constraints are defined by some (fixed) finite automaton describing a regular language R, and the question is, given some DFA A, if A has a synchronizing word from R. This notion explicitly appeared in [4] as an auxiliary tool: it was shown that the synchronization problem of every automaton $A = (\Sigma, Q, \delta)$ whose letters σ have ranks at most r, i.e., $|\delta(Q, \sigma)| \leq r$, is equivalent to the synchronization of an r-state automaton A' under some regular constraints.

The main research question that we look into is to understand for which regular constraints the question of synchronizability is solvable in polynomial time (as it is for $R = \Sigma^*$), or for which it is hard. Furthermore, it would be interesting to see complexity classes different from P and PSPACE to show up (depending on R). In our paper, we give a complete description of the complexity status for constraints that can be described by partial 2-state deterministic automata on alphabets with at most three letters. In this case, indeed, we only observe P and PSPACE situations. However, we also find 3-state automata (on binary input alphabets) that exhibit an NP-complete synchronization problem when considered as constraints. We describe how to generalize our results to larger constraint automata. Moreover, we identify several classes of constraint automata that imply feasible synchronization problems.

This brief outline is an excerpt from [2].

2. Preliminaries and Definitions

Throughout the paper, we consider deterministic finite automata (DFAs). Recall that a DFA A is a tuple $A = (\Sigma, Q, \delta, q_0, F)$, where the alphabet Σ is a finite set of input symbols, Q is the finite state set, with start state $q_0 \in Q$, and final state set $F \subseteq Q$. The transition function $\delta : Q \times \Sigma \to Q$ extends to words from Σ^* and to sets of states $S \subseteq Q$ in the usual way. We sometimes refer to the function δ as a relation and we identify a transition $\delta(q, \sigma) = q'$ with the tuple (q, σ, q'). We call A *complete* if δ is defined for every $(q, a) \in Q \times \Sigma$; if δ is undefined for some (q, a), the automaton A is called *partial*. The set $L(A) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \in F \,\}$ denotes the language

accepted by A. A semi-automaton is a finite automaton without a specified start state and with no specified set of final states. The properties of being *deterministic*, *partial*, and *complete* of semi-automata are defined as for DFA. When the context is clear, we call both deterministic finite automata and semi-automata simply *automata*. We call a deterministic complete semi-automaton a DCSA and a partial deterministic finite automaton a PDFA for short. If we want to add an explicit initial state r and an explicit set of final states S to a DCSA A or change them in a DFA A, we use the notation $A_{r,S}$. For an automaton A over the alphabet Σ, we denote with $A_{\Sigma'}$ for every $\Sigma' \subset \Sigma$ the restriction of A to the alphabet Σ'.

An automaton A is called *synchronizing* if there exists a word $w \in \Sigma^*$ with $|\delta(Q,w)| = 1$. In this case, we call w a *synchronizing word* for A. We call a state $q \in Q$ with $\delta(Q,w) = \{q\}$ for some $w \in \Sigma^*$ a *synchronizing state*. An automaton A is called *returning*, if for every state $q \in Q$, there exists a word $w \in \Sigma^*$ such that $\delta(q,w) = q_0$, where q_0 is the start state of A.

For a fixed PDFA $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$, we define the *constrained synchronization problem*:

Definition 2.1 $L(\mathcal{B})$-CONSTR-SYNC
Input*: DCSA* $A = (\Sigma, Q, \delta)$.
Question*: Is there a synchronizing word* w *for* A *with* $w \in L(\mathcal{B})$*?*

3. Results - Overview

Theorem 3.1 *If* $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ *is returning, then* $L(\mathcal{B})$-CONSTR-SYNC $\in$ *P.*

Theorem 3.2 *Let* $\mathcal{B} = (\Sigma, P, \mu, p_0, F)$ *be a PDFA. Then,* $L(\mathcal{B})$-CONSTR-SYNC $\in$ *NP if there is a* $\sigma \in \Sigma$ *such that for all states* $p \in P$*, if* $L(\mathcal{B}_{p,\{p\}})$ *is infinite, then* $L(\mathcal{B}_{p,\{p\}}) \subseteq \{\sigma\}^*$.

Corollary 3.3 *Under the assumptions of Theorem 3.2,* $L(\mathcal{B})$-CONSTR-SYNC *is in XP with parameter* k *counting the number of sun-structures in* $A_{\{\sigma\}}$ *for an input DCSA* A*.*

Theorem 3.4 *For any two-state binary PDFA* $\mathcal{B}$*,* $L(\mathcal{B})$-CONSTR-SYNC $\in$ *P.*

$a(b+c)^*$	$(a+b)^*c$	$(a+b)^*cc^*$	$a^*b(b+c)^*$
$(a+b+c)(a+b)^*$	$(a+b)^*ca^*$	$a^*b(a+c)^*$	$(a+b)^*c(b+c)^*$
$(a+b)(a+c)^*$	$(a+b)^*c(a+b)^*$	$a^*(b+c)(a+b)^*$	$a^*(b+c)(b+c)^*$

Table 1: Languages accepted by constraint automata (2 states, 3 letters) causing PSPACE-hard synchronization.

Theorem 3.5 *For each constraint language* L *in Table 1 the problem* L-CONSTR-SYNC *is PSPACE-hard. For any other constraint language which can be accepted by a two-state ternary PDFA* $\mathcal{B}$*, the problem* $L(\mathcal{B})$-CONSTR-SYNC *is in P.*

Theorem 3.6 *For* $L = ab^*a$*, the problem* L-CONSTR-SYNC *is NP-complete.*

Theorem 3.7 *Let* $\mathcal{B} = (\Sigma^{\mathcal{B}}, P^{\mathcal{B}}, \mu^{\mathcal{B}}, p_0^{\mathcal{B}}, F)$ *and* $\mathcal{C} = (\Sigma^{\mathcal{C}}, P^{\mathcal{C}}, \mu^{\mathcal{C}}, p_0^{\mathcal{C}}, \emptyset)$ *be PDFAs with* $P^{\mathcal{B}} \cap P^{\mathcal{C}} = \emptyset$*. For* $p_x \in P^{\mathcal{C}}$ *let* $\nu \subseteq \{p_x\} \times (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}) \times \{p_0^{\mathcal{B}}\}$ *define the automaton* $\mathcal{B}' = (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}, P^{\mathcal{B}} \cup P^{\mathcal{C}}, \mu^{\mathcal{B}} \cup \mu^{\mathcal{C}} \cup \nu, p_0^{\mathcal{C}}, F)$*. If the following three conditions are satisfied:*

1. *automaton $\mathcal{B}'$ is deterministic,*
2. *automaton $\mathcal{C}' = (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}, P^{\mathcal{C}} \cup \{p_0^{\mathcal{B}}\}, \mu^{\mathcal{C}} \cup \nu \cup \{p_0^{\mathcal{B}}\} \times (\Sigma^{\mathcal{B}} \cup \Sigma^{\mathcal{C}}) \times \{p_0^{\mathcal{B}}\})$ is carefully synchronizing, and*
3. *there exists a synchronizing word $v = v_1 \dots v_n$ for $\mathcal{C}'$ such that $v_1 \dots v_{n-1} \in L(\mathcal{C}_{p_x})$, where $\mathcal{C}_{p_x}$ results from $\mathcal{C}$ by adding p_x to the set of final states,*

then $L(\mathcal{B})$-CONSTR-SYNC $\leq L(\mathcal{B}')$-CONSTR-SYNC.

Corollary 3.8 *Let the language-family $\mathcal{L}$ consists of languages $L_i := (b^*a)^i$ with $i \geq 2$. The constrained synchronization problem for all languages in $\mathcal{L}$ is NP-complete.*

References

[1] J. ČERNÝ, Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis* **14** (1964) 3, 208–216.

[2] H. FERNAU, V. V. GUSEV, S. HOFFMANN, M. HOLZER, M. V. VOLKOV, P. WOLF, Computational Complexity of Synchronization under Regular Constraints. In: *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany.*. 2019, 63:1–63:14.

[3] Z. GAZDAG, S. IVÁN, J. NAGY-GYÖRGY, Improved upper bounds on synchronizing nondeterministic automata. *Information Processing Letters* **109** (2009) 17, 986–990.

[4] V. V. GUSEV, Synchronizing Automata of Bounded Rank. In: N. MOREIRA, R. REIS (eds.), *Implementation and Application of Automata - 17th International Conference, CIAA*. LNCS 7381, Springer, 2012, 171–179.

[5] P. MARTYUGIN, Complexity of Problems Concerning Reset Words for Some Partial Cases of Automata. *Acta Cybernetica* **19** (2009) 2, 517–536.

[6] P. V. MARTYUGIN, Computational Complexity of Certain Problems Related to Carefully Synchronizing Words for Partial Automata and Directing Words for Nondeterministic Automata. *Theory of Computing Systems* **54** (2014) 2, 293–304.

[7] I. K. RYSTSOV, Polynomial Complete Problems in Automata Theory. *Information Processing Letters* **16** (1983) 3, 147–151.

[8] S. SANDBERG, Homing and Synchronizing Sequences. In: M. BROY, B. JONSSON, J. KATOEN, M. LEUCKER, A. PRETSCHNER (eds.), *Model-Based Testing of Reactive Systems, Advanced Lectures*. LNCS 3472, Springer, 2005, 5–33.

[9] M. SZYKUŁA, Improving the Upper Bound on the Length of the Shortest Reset Word. In: R. NIEDERMEIER, B. VALLÉE (eds.), *35th Symposium on Theoretical Aspects of Computer Science, STACS*. LIPIcs 96, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 56:1–56:13.

[10] M. V. VOLKOV, Synchronizing Automata and the Černý Conjecture. In: C. MARTÍN-VIDE, F. OTTO, H. FERNAU (eds.), *Language and Automata Theory and Applications, Second International Conference, LATA*. LNCS 5196, Springer, 2008, 11–27.

THEORIE-AG 2019

S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 42–44.

Universität Bremen

Eigenschaften und Zustandskomplexität kommutativer regulärer Sprachen

Stefan Hoffmann[(A)]

[(A)]Universität Trier, Abteilung Informatikwissenschaften, CIRT, 54296 Trier
{hoffmanns}@uni-trier.de

Zusammenfassung

Eine formale Sprache heisst kommutativ, sofern sie unter dem vertauschen einzelner Zeichen abgeschlossen ist. Ein endlicher Automat heisst kommutativ, wenn die durch die Zeichen induzierten Abbildungen kommutieren. Es hat sich gezeigt dass kommutative Automaten in Härtebeweisen auftreten können, demgegenüber aber eine einfache kombinatorische Struktur aufweisen. Wir werden ein kanonisches Automatenmodell für diese Sprachen einführen welches sich als nützlich zur Herleitung der Zustandskomplexität verschiedener Operationen erwiesen hat. Dabei zeigt sich dass kommutative reguläre Sprachen sich bezüglich Zustandskomplexität unter der Shuffle-Operation und der Projektion besser verhalten als im allgemeinen Fall. Weiterhin sind minimale synchronisierende Wörter für kommutative Automaten stets linear beschränkt. Wir stellen einen kombinatorischen Beweis dieser Aussage vor, und klassifizieren außerdem jene kommutativen Automaten für die diese Schranke angenommen wird.

1. Einleitung

Ein *deterministischer endlicher Halbautomat* (DEH) wird beschrieben durch ein Tripel $A = (S, \Sigma, \delta)$, wobei S, Σ Alphabete sind und $\delta : S \times \Sigma \to S$ eine Abbildung ist, die in natürlicher Weise auch als $\delta : 2^S \times \Sigma^* \to 2^S$ aufgefasst werden kann. Ein Wort $w \in \Sigma^*$ heißt *synchronisierend*, falls $|\delta(S, w)| = 1$. Ein *endlicher Automat* (DEA) wird beschrieben durch ein Tupel $A = (S, \Sigma, \delta, s_0, F)$, wobei (S, Σ, δ) ein DEH bildet, sowie $F \subseteq S$ die Finalzustandsmenge darstellt und s_0 der Startzustand ist. Es ist $L(A) = \{w \in \Sigma^* : \delta(s_0, w) \in F\}$ die durch den Automaten beschriebene Sprache. Ein DEH (bzw. ein DEA) heisst *kommutativ* sofern für alle Zustände $\in S$ und alle Zeichen $a, b \in \Sigma$ gilt $\delta(s, ab) = \delta(s, ba)$. Eine Sprache $L \subseteq \Sigma^*$ heisst *kommutativ* sofern aus $uabv \in L$ folgt $ubav \in L$ für $u, v \in \Sigma^*$ und $a, b \in \Sigma$. Eine reguläre Sprache ist genau dann kommutativ wenn Sie von einem kommutativen Automaten akzeptiert wird. Es sei $\Sigma = \{a_1, \ldots, a_k\}$ unser Alphabet. Die Homomorphismen $\pi_i : \Sigma^* \to \{a_i\}$ für $i = 1, \ldots, k$, welche durch $\pi_i(a_i) = a_i$ und $\pi_i(a_j) = \varepsilon$ für $i \neq j$ gegeben sind heissen *Projektionen*. Weiterhin ist zu $U, V \subseteq \Sigma^*$ die Operation

$$U \sqcup\!\sqcup V := \{u_1 v_1 u_2 v_2 \cdots u_n v_n : u_1 u_2 \cdots u_n \in U, v_1 v_2 \cdots v_n \in V, u_i, v_i \in \Sigma^*, i = 1, \ldots, n\}$$

die *Shuffle*-Operation auf Sprachen. Der minimale, kanonische kommutative Automat wurde in [3] eingeführt.

Definition 1.1 *Es sei L eine kommutative Sprache. Der minimale kanonische kommutative Automat ist $\mathcal{C}_L = (\Sigma, S_1 \times \ldots \times S_k, \delta, s_0, F)$ mit*

$$\begin{aligned} S_j &:= \{[a_j^m]_{\equiv_L} : m \geq 0\}, \\ s_0 &:= ([\varepsilon]_{\equiv_L}, \ldots, [\varepsilon]_{\equiv_L}), \\ F &:= \{([\pi_1(w)]_L, \ldots, [\pi_k(w)]_L) : w \in L\} \end{aligned}$$

und $\delta((s_1, \ldots, s_j, \ldots, s_k), a_j) := (s_1, \ldots, \delta_j(s_j, a_j), \ldots, s_k)$ mit Ein-Zeichen-Übergängen

$$\delta_j([a_j^m]_{\equiv_L}, a_j) := [a_j^{m+1}]_{\equiv_L}$$

für $j = 1, \ldots, k$ und $s_0 := ([\varepsilon]_{\equiv_L}, \ldots, [\varepsilon]_{\equiv_L})$.

2. Ergebnisse zur Zustandskomplexität

Die Zustandskomplexität einer Operation auf regulären Sprachen ist die maximale Größe des kleinsten Automaten welcher die Ergebnissprache akzeptiert in Abhängigkeit von den Zustandsgrößen der Eingabeautomaten, siehe [6] für weitere Informationen. Sind zwei endliche Automaten A und B mit m und n Zuständen gegeben, so wurde in [1] die allgemeine Schranke $2^{mn-1} + 2^{(m-1)(n-1)}(2^{m-1} - 1)(2^{n-1} - 1)$ für die Zustände eines Automaten welcher $L(A) \sqcup\!\sqcup L(B)$ akzeptiert gezeigt.

Theorem 2.1 *Sind U und V kommutative reguläre Sprachen, welche durch Automaten mit n und m Zuständen akzeptiert werden, so hat ein kleinster Automat welche $U \sqcup\!\sqcup V$ akzeptiert höchstens $(2nm)^k$ Zustände.*

Die Zustandszahl eines Automaten für $\pi_i(L)$ wächst im Allgemeinen mit $e^{\Theta(\sqrt{n \ln n})}$.

Theorem 2.2 *Ist U kommutativ und regulär und wird durch einen Automaten mit n Zuständen akzeptiert, so kann $\pi_i(U)$ für $i = 1, \ldots, k$ von einem Automaten mit höchstens n Zuständen akzeptiert werden.*

Diese Ergebnisse finden sich in [4].

3. Ergebnisse zur Synchronisierbarkeit

In [5] wurde gezeigt das in einem kommutativen DEH mit n Zuständen ein kürzestes synchronisierendes Wort höchstens Länge $n-1$ haben kann. Der Beweis wurde durch Linearisierung der Automaten mit algebraischen Methoden erbracht. Der Autor fragte abschließend nach einem kombinatorischen Beweis. Dieser wurde in [2] geliefert, darüber hinaus konnten jene Automaten klassifiert werden für welche die Schranke angenommen wird. Ein kommutativer synchronisierender Automat heisst *extremal*, falls das kürzeste synchronisierende Wort die Länge $n-1$ hat. Zu einem DEA $A = (S, \Sigma, \delta, s_0, F)$ (oder DEH $A = (S, \Sigma, \delta)$) und $T \subseteq S$ bezeichnen wir mit $A_{|T} = (S, \Sigma, \delta_{|T})$ den partiellen DEH (d.h. $\delta_{|T} : T \times \Sigma \to T$ ist nur eine partielle Funktion) mit $\delta_{|T}(t, x) = \delta(t, x)$ falls $t \in T$, $x \in \Sigma$ und $\delta(t, x) \in T$, anderenfalls undefiniert.

Lemma 3.1 *In einem synchronisierenden kommutativen vollständigen DEH gibt es nur einen eindeutig bestimmten synchronisierenden Zustand, und dieser ist ein Fangzustand (d.h. alle ausgehenden Transitionen sind Schleifen).*

Zu einem partiellen DEH $A = (S, \Sigma, \delta)$ sagen wir dass $a \in \Sigma$ als Verschiebung auf S operiert, sofern eine lineare Ordnung auf den Zuständen existiert so dass für alle Zustände s außer dem maximalen Element s_f in dieser Ordnung $\delta(s, a)$ das kleinste Element ist, welches größer als s ist.

Theorem 3.2 *Es sei $\Sigma = \{a_1, \ldots, a_k\}$ und $A = (S, \Sigma, \delta)$ ein vollständiger, kommutativer und synchronisierbarer extremaler DEH mit eindeutigen synchronisierenden Zustand s_f. Sofern alle Buchstaben unterschiedlich operieren, dann kann man $S = S_1 \cup \cdots \cup S_k$ schreiben mit $S_1 \cap \cdots \cap S_k = \{s_f\}$ und jedes $A_{|S_i}$ ist ein vollständiger Automat auf welchem a_i als Verschiebung operiert, wobei s_f der maximale Zustand ist, und jedes andere Symbol als identische Abbildung operiert.*

Corollary 3.3 *Jeder extremale, vollständige, synchronisierbare kommutative Automat ist geordnet. Im Falle eines binären Alphabets ist er sogar linear geordnet.*

Literatur

[1] J. A. BRZOZOWSKI, G. JIRÁSKOVÁ, B. LIU, A. RAJASEKARAN, M. SZYKULA, On the State Complexity of the Shuffle of Regular Languages. In: *Descriptional Complexity of Formal Systems - 18th IFIP WG 1.2 International Conference, DCFS 2016, Bucharest, Romania, July 5-8, 2016. Proceedings*. 2016, 73–86.
https://doi.org/10.1007/978-3-319-41114-9_6

[2] H. FERNAU, S. HOFFMANN, Extensions to minimal synchronizing words. *Journal of Automata, Languages and Combinatorics* **24** (2019), 287–307.

[3] A. C. GÓMEZ, G. I. ALVAREZ, Learning Commutative Regular Languages. In: *Grammatical Inference: Algorithms and Applications, 9th International Colloquium, ICGI 2008, Saint-Malo, France, September 22-24, 2008, Proceedings*. 2008, 71–83.
https://doi.org/10.1007/978-3-540-88009-7_6

[4] S. HOFFMANN, Commutative Regular Languages - Properties and State Complexity. In: M. CIRIC, M. DROSTE, J. PIN (eds.), *Algebraic Informatics - 8th International Conference, CAI 2019, Niš, Serbia, June 30 - July 4, 2019, Proceedings*. Lecture Notes in Computer Science 11545, Springer, 2019, 151–163.
https://doi.org/10.1007/978-3-030-21363-3_13

[5] I. K. RYSTSOV, Exact linear bound for the length of reset words in commutative automata. *Publicationes Mathematicae, Debrecen* **48** (1996) 3-4, 405–409.

[6] S. YU, Q. ZHUANG, K. SALOMAA, The State Complexities of Some Basic Operations on Regular Languages. *Theoretical Computer Science* **125** (1994) 2, 315–328.
http://dx.doi.org/10.1016/0304-3975(92)00011-F

THEORIE-AG 2019

S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 45–48.

Universität Bremen

Graph and String Parameters: Connections Between Pathwidth, Cutwidth and the Locality Number

Katrin Casel[(A)] Joel D. Day[(B)] Pamela Fleischmann[(C)]
Tomasz Kociumaka[(D)] Florin Manea[(C)] Markus L. Schmid[(E)]

[(A)]Hasso Plattner Institute, University of Potsdam, Germany
Katrin.Casel@hpi.de

[(B)]Loughborough University, UK
J.Day@lboro.ac.uk

[(C)]Kiel University, Germany
{fpa,flm}@informatik.uni-kiel.de

[(D)]Bar-Ilan University, Israel, and University of Warsaw, Poland
kociumaka@mimuw.edu.pl

[(E)]Humboldt University, Berlin, Germany
mlschmid@mlschmid.de

Abstract

We investigate the locality number, a recently introduced structural parameter for strings (with applications in pattern matching with variables), and its connection to two important graph-parameters, cutwidth and pathwidth. These connections allow us to show that computing the locality number is NP-hard but fixed parameter tractable (when the locality number or the alphabet size is treated as a parameter), and can be approximated with ratio $O(\sqrt{\log \mathsf{opt}} \log n)$. As a by-product, we also relate cutwidth via the locality number to pathwidth, which is of independent interest, since it improves the currently best known approximation algorithm for cutwidth. In addition to these main results, we also consider the possibility of greedy-based approximation algorithms for the locality number.

1. Overview

Graphs, on the one hand, and strings, on the other, are two different types of data objects and they have certain particularities. Graphs seem to be more popular in fields like classical and parameterised algorithms and complexity (due to the fact that many natural graph problems are intractable), while fields like formal languages, pattern matching, verification or compression are more concerned with strings. Moreover, both the field of graph algorithms as well as string algorithms are well established and provide rich toolboxes of algorithmic techniques, but they differ in that the former is tailored to computationally hard problems (e. g., the approach of treewidth and related parameters), while the latter focuses on providing efficient data-structures

for near-linear-time algorithms. Nevertheless, it is sometimes possible to bridge this divide, i. e., by "flattening" a graph into a sequential form, or by "inflating" a string into a graph, to make use of respective algorithmic techniques otherwise not applicable. This paradigm shift may provide the necessary leverage for new algorithmic approaches.

In this presentation, we are concerned with certain structural parameters (and the problems of computing them) for graphs and strings: the *cutwidth* $\mathsf{cw}(G)$ of a graph G (i. e., the maximum number of "stacked" edges if the vertices of a graph are drawn on a straight line), the *pathwidth* $\mathsf{pw}(G)$ of a graph G (i. e., the minimum width of a tree decomposition the tree structure of which is a path), and the *locality number* $\mathsf{loc}(\alpha)$ of a string α (explained in more detail in the next paragraph). By CUTWIDTH, PATHWIDTH and LOC, we denote the corresponding decision problems and with the prefix MIN, we refer to the minimisation variants. The two former graph-parameters are very classical. Pathwidth is a simple (yet still hard to compute) subvariant of treewidth, which measures how much a graph resembles a path. The problems PATHWIDTH and MINPATHWIDTH are intensively studied (in terms of exact, parameterised and approximation algorithms) and have numerous applications (see the surveys and textbook [4, 12, 2]). CUTWIDTH is the best known example of a whole class of so-called *graph layout problems* (see the survey [8, 14] for detailed information), which are studied since the 1970s and were originally motivated by questions of circuit layouts.

The locality number is rather new and we shall discuss it in more detail. A word is k-local if there exists an order of its symbols such that, if we *mark* the symbols in the respective order (which is called a *marking sequence*), at each stage there are at most k contiguous blocks of marked symbols in the word. This k is called the *marking number* of that marking sequence. The *locality number* of a word is the smallest k for which that word is k-local, or, in other words, the minimum marking number over all marking sequences. For example, the marking sequence $\sigma = (\mathtt{x},\mathtt{y},\mathtt{z})$ marks $\alpha = \mathtt{xyxyzxz}$ as follows (marked blocks are illustrated by overlines): $\mathtt{xyxyzxz}$, $\overline{\mathtt{x}}\mathtt{y}\overline{\mathtt{x}}\mathtt{yz}\overline{\mathtt{x}}\mathtt{z}$, $\overline{\mathtt{xyxy}}\mathtt{z}\overline{\mathtt{x}}\mathtt{z}$, $\overline{\mathtt{xyxyzxz}}$; thus, the marking number of σ is 3. In fact, all marking sequences for α have a marking number of 3, except $(\mathtt{y},\mathtt{x},\mathtt{z})$, for which it is 2: $\mathtt{x}\overline{\mathtt{y}}\mathtt{x}\overline{\mathtt{y}}\mathtt{zxz}$, $\overline{\mathtt{xyxy}}\mathtt{z}\overline{\mathtt{x}}\mathtt{z}$, $\overline{\mathtt{xyxyzxz}}$. Thus, the locality number of α, denoted by $\mathsf{loc}(\alpha)$, is 2.

The locality number has applications in pattern matching with variables [6]. A *pattern* is a word that consists of *terminal symbols* (e. g., $\mathtt{a},\mathtt{b},\mathtt{c}$), treated as constants, and *variables* (e. g., $x_1, x_2, x_3, \ldots$). A pattern is mapped to a word by substituting the variables by strings of terminals. For example, $x_1x_1\mathtt{bab}x_2x_2$ can be mapped to $\mathtt{acacbabcc}$ by the substitution $(x_1 \to \mathtt{ac}, x_2 \to \mathtt{c})$. Deciding whether a given pattern matches (i. e., can be mapped to) a given word (called the *matching problem*) is one of the most important problems that arise in the study of patterns but, unfortunately, it is NP-complete [1] in general and also intractable in the parameterised setting [11]). As demonstrated in [15], for the matching problem a paradigm shift as mentioned above yields a very promising algorithmic approach. More precisely, any class of patterns with bounded treewidth (for suitable graph representations) can be matched in polynomial-time. However, computing (and therefore algorithmically exploiting) the treewidth of a pattern is difficult (see the discussion in [10, 15]), which motivates more direct string-parameters that bound the treewidth and are simple to compute (virtually all known structural parameters that lead to tractability, see, e.g., [10] and the references therein, are of this kind (the efficiently matchable classes investigated in [7] are one of the rare exceptions)). This also establishes an interesting connection between ad-hoc string parameters and the more general (and much better studied) graph parameter treewidth. The locality number is a simple parameter

directly defined on strings, it bounds the treewidth and the corresponding marking sequences can be seen as instructions for a dynamic programming algorithm. However, compared to other "tractability-parameters", it seems to cover best the treewidth of a string, but whether it can be efficiently computed is unclear.

In this presentation, we investigate the problem of computing the locality number and, by doing so, we establish an interesting connection to the graph parameters cutwidth and pathwidth with algorithmic implications for approximating cutwidth. In the following, we first discuss related results in more detail and then outline our respective contributions.

Known Results and Open Questions: For LOC, only exact exponential-time algorithms are known and whether it can be solved in polynomial-time, or whether it is at least fixed-parameter tractable is mentioned as open problems in [6]. Approximation algorithms have not yet been considered. Addressing these questions is the main purpose of this presentation.

PATHWIDTH and CUTWIDTH are NP-complete, but fixed-parameter tractable with respect to parameter $\mathsf{pw}(G)$ or $\mathsf{cw}(G)$, respectively (even with "linear" fpt-time $g(k)\,\mathrm{O}(n)$ [3, 5, 16]). With respect to approximation, their minimisation variants have received a lot of attention, mainly because they yield (like many other graph parameters) general algorithmic approaches for numerous graph problems, i. e., a good linear arrangement or path-decomposition can often be used for a dynamic programming (or even divide and conquer) algorithm. More generally speaking, pathwidth and cutwidth are related to the more fundamental concepts of small balanced vertex or edge separators for graphs (i. e., a small set of vertices (or edges, respectively) that, if removed, divides the graph into two parts of roughly the same size. More precisely, $\mathsf{pw}(G)$ and $\mathsf{cw}(G)$ are upper bounds for the smallest balanced *vertex* separator of G and the smallest balanced *edge* separator of G, respectively (see [9] for further details and explanations of the algorithmic relevance of balanced separators). The best known approximation algorithms for MINPATHWIDTH and MINCUTWIDTH (with approximations ratios of $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(n))$ and $\mathrm{O}(\log^2(n))$, respectively) follow from approximations of vertex separators (see [9]) and edge separators (see [13]), respectively.

Our Contributions: There are two natural approaches to represent a word α over alphabet Σ as a graph $G_\alpha = (V_\alpha, E_\alpha)$: (1) $V_\alpha = \{1, 2, \ldots, |\alpha|\}$ and the edges are somehow used to represent the actual symbols, or (2) $V_\alpha = \Sigma$ and the edges are somehow used to represent the positions of α. We present a reduction of type (2) such that $|E_\alpha| = \mathrm{O}(|\alpha|)$ and $\mathsf{cw}(G_\alpha) = 2\,\mathsf{loc}(\alpha)$, and a reduction of type (1) such that $|E_\alpha| = \mathrm{O}(|\alpha|^2)$ and $\mathsf{loc}(\alpha) \leq \mathsf{pw}(G_\alpha) \leq 2\,\mathsf{loc}(\alpha)$. Since these reductions are parameterised reductions and also allow to transfer approximation results, we conclude that LOC is fixed-parameter tractable if parameterised by $|\Sigma|$ or by the locality number (answering the respective open problem from [6]), and also that there is a polynomial-time $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(n))$-approximation algorithm for MINLOC.

In addition, we also show a way to represent an arbitrary multi-graph $G = (V, E)$ by a word α_G over alphabet V, of length $|E|$ and with $\mathsf{cw}(G) = \mathsf{loc}(\alpha)$. This describes a Turing-reduction from CUTWIDTH to LOC which also allows to transfer approximation results between the minimisation variants. As a result, we can conclude that LOC is NP-complete (which solves the other open problem from [6]). Finally, by plugging together the reductions from MINCUTWIDTH to MINLOC and from MINLOC to MINPATHWIDTH, we obtain a reduction which transfers approximation results from MINPATHWIDTH to MINCUTWIDTH, which yields an $\mathrm{O}(\sqrt{\log(\mathsf{opt})}\log(n))$-approximation algorithm for MINCUTWIDTH. This improves, to our knowledge for the first time since 1999, the best approximation for CUTWIDTH from [13]. The

paper we present here was published in Proc. of ICALP 2019 and it is available on Arxiv.

References

[1] D. ANGLUIN, Finding patterns common to a set of strings. *J. Comput. Syst. Sci.* **21** (1980), 46–62.

[2] H. L. BODLAENDER, A Tourist Guide through Treewidth. *Acta Cybern.* **11** (1993) 1-2, 1–21. `http://www.inf.u-szeged.hu/actacybernetica/edb/vol11n1_2/pdf/Bodlaender_1993_ActaCybernetica.pdf`

[3] H. L. BODLAENDER, A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM J. Comput.* **25** (1996) 5, 1305–1317.

[4] H. L. BODLAENDER, A Partial k-Arboretum of Graphs with Bounded Treewidth. *Theor. Comput. Sci.* **209** (1998) 1-2, 1–45.

[5] H. L. BODLAENDER, Fixed-Parameter Tractability of Treewidth and Pathwidth. In: H. L. BODLAENDER, R. DOWNEY, F. V. FOMIN, D. MARX (eds.), *The Multivariate Algorithmic Revolution and Beyond*. LNCS 7370, 2012, 196–227.

[6] J. D. DAY, P. FLEISCHMANN, F. MANEA, D. NOWOTKA, Local Patterns. In: S. V. LOKAM, R. RAMANUJAM (eds.), *Proc. FSTTCS*. LIPIcs 93, 2017, 24:1–24:14.

[7] J. D. DAY, P. FLEISCHMANN, F. MANEA, D. NOWOTKA, M. L. SCHMID, On Matching Generalised Repetitive Patterns. In: *Proc. DLT*. LNCS, 2018, 269–281.

[8] J. DÍAZ, J. PETIT, M. SERNA, A Survey of Graph Layout Problems. *ACM Comput. Surv.* **34** (2002) 3, 313–356.

[9] U. FEIGE, M. HAJIAGHAYI, J. R. LEE, Improved Approximation Algorithms for Minimum Weight Vertex Separators. *SIAM J. Comput.* **38** (2008) 2, 629–657.

[10] H. FERNAU, F. MANEA, R. MERCAŞ, M. L. SCHMID, Pattern Matching with Variables: Fast Algorithms and New Hardness Results. In: E. W. MAYR, N. OLLINGER (eds.), *Proc. STACS*. LIPIcs 30, 2015, 302–315.

[11] H. FERNAU, M. L. SCHMID, Y. VILLANGER, On the Parameterised Complexity of String Morphism Problems. *Theory Comput. Syst.* **59** (2016) 1, 24–51.

[12] T. KLOKS, *Treewidth, Computations and Approximations*. Lecture Notes in Computer Science 842, Springer, 1994.

[13] T. LEIGHTON, S. RAO, Multicommodity Max-flow Min-cut Theorems and Their Use in Designing Approximation Algorithms. *J. ACM* **46** (1999) 6, 787–832.

[14] J. PETIT, Addenda to the Survey of Layout Problems. *Bulletin of the EATCS* **105** (2011), 177–201. `http://eatcs.org/beatcs/index.php/beatcs/article/view/98`

[15] D. REIDENBACH, M. L. SCHMID, Patterns with bounded Treewidth. *Inf. Comput.* **239** (2014), 87–99.

[16] D. M. THILIKOS, M. J. SERNA, H. L. BODLAENDER, Cutwidth I: A linear time fixed parameter algorithm. *J. Algorithms* **56** (2005) 1, 1–24.

THEORIE-AG 2019

S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 49–51.

Universität Bremen

Separating Languages of Infinite Words in the Mostowski Hierarchy

Christopher Hugenroth[(A)]

[(A)] christopher.hugenroth@tu-ilmenau.de

Abstract

The Mostowski hierarchy classifies ω-languages according to the range of priorities required to recognize them with a deterministic parity automaton. The membership problem for the Mostowski hierarchy can be solved efficiently in many cases. We generalize this result and show that for two ω-regular languages a separating language can be determined in polynomial time and that this language is minimal with respect to the Mostowski hierarchy.

1. Introduction

A deterministic parity automaton (DPA) is a finite state automaton with priorities assigned to each state. A DPA accepts an ω-word if the maximal priority seen in its unique, infinite run is even. An ω-language L is contained in the Mostowski class $[i, j]$ if there is a parity automaton $\mathcal{A}$ with priorities in $\{i, \ldots, j\}$ and $L(\mathcal{A}) = L$. The priorities of a DPA can be assumed to be $\{0, \ldots, j\}$ or $\{1, \ldots, j\}$ and we can restrict ourselves to the corresponding Mostowski classes, see figure 1. This hierarchy is of interest because it is a good indicator of the complexity of an ω-language. The Mostowski membership problem is to determine whether a language L is contained in a certain Mostowski class.

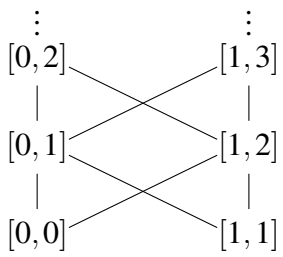


Figure 1: The classes of the Mostowski hierarchy and their relations.

The membership problem can be solved for many classes of automata and the corresponding classes of recognized languages with the so-called pattern method. This method searches for

[(A)] The results are part of my master thesis at RWTH Aachen. The thesis was supervised by Christof LÃűding.

a certain pattern in an automaton that recognizes a language L and thereby determines the minimal Mostowski class of L. For deterministic word automata it suffices to determine the maximal nesting of loops in the automaton, [1], [5]. For deterministic tree automata one can determine the class in the *nondeterministic* Mostowski hierarchy, [2]. The patterns used here are called flowers.

In this paper, we study the Mostowski separation problem which is to decide whether two given ω-regular languages L_1, L_2 can be separated with a language in $[i, j]$. A language L separates L_1 and L_2 if $L_1 \subseteq L$ and $L_2 \cap L = \emptyset$. Notice that L_1 and L_2 can be separated iff they are disjoint.

A language L is in $[i, j]$ if, and only if, L and its complement can be separated with a language from $[i, j]$, so membership can be reduced to separation. However, the converse doesn't seem to hold since for membership there is a fixed recognition device but separation admits infinitely many candidates, as noted in [3]. The main idea of our paper is to use product automata as a fixed device and to solve separation with the pattern-method.

We show that it suffices to compute the maximal nesting depth of loops in a special product automaton $\mathcal{A}_1 \times \mathcal{A}_2$ to solve separation for $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$. We can use the product automaton to obtain a separator if the languages are disjoint. Further, we show that the nesting depth of loops in $\mathcal{A}_1 \times \mathcal{A}_2$ is minimal among all separators. For this we use arguments similar to the ones used in the membership case, [2]. Thus, the obtained separator has a minimal nesting depth and therefore is in the minimal Mostowski class that contains a separator.

Theorem 1.1 *Let L_1, L_2 be two disjoint ω-regular languages given as DMA.*
A DMA $\mathcal{A}$ whose language separates L_1, L_2 can be computed in polynomial time. Further, $L(\mathcal{A})$ is minimal among all separating languages with respect to the Mostowski hierarchy.

For deterministic parity automata a similar result holds but it remains open whether a separator can be computed in polynomial time.

Corollary 1.2 *Let L_1, L_2 be two disjoint ω-regular languages given as DPA.*
The minimal Mostowski class that contains a language separating L_1 and L_2 can be computed in polynomial time.

I thank Christof LÃűding, the supervisor of my Master's thesis, for proposing the topic and for his comments.

References

[1] O. Carton, R. Maceiras *Computing the Rabin index of a parity automaton* in Theoretical Informatics and Applications 33, 495–505 (1999)

[2] D. Niwiński, I. Walukiewicz *Deciding Nondeterministic Hierarchy of Deterministic Tree Automata*, Electronic Notes in Theoretical Computer Science 123, 195-208 (2005)

[3] T. Place, M. Zeitoun *Separating Languages with First Orer Logic*, Logical Methods in Computer Science Vol. 12(1:5), pp. 1âĂŞ30 (2016)

[4] L. Staiger, K. W. Wagner,*Automatentheoretische und automatenfreie Characterisierungen topologischer Klassen regulÃd'rer Folgemengen*, Elektronische Informationsverarbeitung und Kybernetik EIK, 10 379âĂŞ392 (1974)

[5] K. W. Wagner *On ω-regular sets* in Information and Control 43, 123–177 (1979)

THEORIE-AG 2019

S. Maneth (Herausgeber): Theorietag 2019, Bremen, 25. – 27.9.2019
Universität Bremen, Technischer Bericht der AG Datenbanken, S. 52–56.

Universität Bremen

Regular Expressions with Backreferences: Polynomial-Time Matching Techniques

Markus L. Schmid[(A)]

[(A)]Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany
MLSchmid@MLSchmid.de

Abstract

Regular expressions with backreferences (regex, for short), as supported by most modern libraries for regular expression matching, have an NP-complete matching problem. We define a complexity parameter of regex, called active variable degree, such that regex with this parameter bounded by a constant can be matched in polynomial-time. Moreover, we formulate a novel type of determinism for regex (on an automaton-theoretic level), which yields the class of memory-deterministic regex that can be matched in time $O(|w|p(|\alpha|))$ for a polynomial p (where α is the regex and w the word). It is also shown that natural extensions of these concepts lead to properties of regex that are intractable to check.

1. Regular Expressions with Backreferences

Regular expressions have been first introduced by Kleene in 1956 [7] as a theoretical concept (an early implementation is due to Thompson [11]). Since then, they have been enriched with practically motivated extensions and modifications, which is mainly due to their rather high practical relevance.

Regular expressions have excellent decidability- and complexity-properties, while at the same time providing sufficient expressive power for many important computational tasks. The practical enhancements added over the years are mostly "syntactic sugar" (e. g., character-groups), or are even restrictions (as, e. g., in deterministic regular expressions) and therefore preserve these positive properties. However, adding so-called *backreferences* drastically increases expressive power and therefore leads to intractability and even undecidability. We shall explain this concept on an intuitive level. A successful *match* of a regular expression r to a word w can be interpreted as mapping the symbols of the word to the symbols of the expression. For example, a possible match of $r = (\mathtt{a} \vee \mathtt{b})^*\mathtt{d}$ to $\mathtt{bbd}$, maps the two occurrences of $\mathtt{b}$ to the occurrence of $\mathtt{b}$ in r, and the occurrence of $\mathtt{d}$ to the occurrence of $\mathtt{d}$ in r. Implicitly, this also maps larger factors to subexpressions of r, e. g., prefix $\mathtt{bb}$ is mapped to $(\mathtt{a} \vee \mathtt{b})^*$. A backreference now allows to postulate that such a factor previously allocated to a subexpression should be repeated in the word to be matched. For example, the $x\{\ldots\}$-construct in the expression $s = x\{(\mathtt{a} \vee \mathtt{b})^*\}\mathtt{d}x$ stores in variable x whatever subword is matched to the subexpression $(\mathtt{a} \vee \mathtt{b})^*$, and the following occurrence of variable x then refers to exactly this subword (thus,

s describes the non-regular language $\{w\mathtt{d}w \mid w \in \{\mathtt{a},\mathtt{b}\}^*\}$). We use the term *regex* for regular expressions with such backreferences. Next, we discuss a few examples to build up some intuition.

The regex s from above and the language it describes, although non-regular, are rather simple. However, nesting several backreferenced subexpressions, or subjecting them to the star operator, e. g., $t = y\{(x\mathtt{b}x\{\mathtt{a}^*\}\mathtt{b}x)^+\}\mathtt{d}y^*x$, makes the described language much more involved. Moreover, we encounter semantical particularities that are not explained on this intuitive level: What is the value of variable x if its "definition" $x\{\ldots\}$ is repeated several times under a star? What does an occurrence of x refer to if it is encountered before it has been defined for the first time? Other noteworthy examples of (unary) regex are $(x\{y\}y\{x\mathtt{a}\})^*$ and $x\{\mathtt{aa}^+\}x^+$, which describe the sets $\{\mathtt{a}^{n^2} \mid n \geq 0\}$ of square numbers and the set $\{\mathtt{a}^{mn} \mid n, m \geq 2\}$ of composite numbers (in unary encoding), respectively. The simple context-free language $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$ cannot be expressed by a regex.

Adding backreferences to regular expressions has severe negative consequences with respect to decidability and computational complexity. The *matching problem*, i. e., deciding whether a given regex can match a given word, is NP-complete (even for strongly restricted variants) [1, 3, 4], and decision problems like inclusion, equivalence and universality are undecidable [5] (even if the input expressions only use one variable with only a bounded number of occurrences). Nevertheless, regular expression libraries of almost all modern programming languages (like, e. g., Java, PERL, Python and .NET) support backreferences (although they syntactically and even semantically slightly differ from each other (see the discussion in [6])).

The general NP-completeness of the matching problem was shown in [1], but also follows from matching *patterns with variables* [2], i. e., checking whether the variables x_i in a pattern $\alpha \in (\Sigma \cup \{x_i \mid i \in \mathbb{N}\})^*$ can be uniformly replaced by words from Σ^* in order to obtain a given word (see [8] for a recent survey). Since patterns describe a proper subclass of the class of regex languages, hardness results (see, e. g., [3, 4]) directly carry over to regex (e. g., the matching problem for regex is W[1]-hard, even if parameterised by the size of the regex or the size of the word). Consequently, patterns are a quite successful tool for obtaining negative results for regex, but the many known positive algorithmic approaches to matching patterns (see [9]) are tailored to the "backreferencing-aspect" and seem unfit for handling the "regular expression-aspect" of regex. In fact, even though there are many deep theoretical (yet negative) results about the complexity and decidability of regex, positive algorithmic approaches are rather scarce.

2. Memory Automata

A suitable algorithmic framework for regex are *memory finite automata* (MFA for short), which have been first introduced in [10]. Syntactically, an MFA(k) (k is the number of memories) over an alphabet Σ is just an NFA with transition labels from Σ and $\{\mathtt{o}(i), \mathtt{c}(i), i \mid 1 \leq i \leq k\}$. Σ-transitions have the usual meaning, whereas labels $\mathtt{o}(i)$ and $\mathtt{c}(i)$ are non-reading and stand for "open memory i" and "close memory i", respectively. In a computation, each open memory will record the consumed input, while closed memories do nothing. Whenever a closed memory is opened again, it will lose its previous content. The special transition labels i, $1 \leq i \leq k$, *recall* memory i, which means that the content of the memory is consumed from the input in one step. By using memories to simulate backreferences, regex can be transformed into equivalent

MFA. For example, $x\{(\mathtt{a}\vee\mathtt{b})^*\}\mathtt{d}x$ from above can be simulated by an MFA(1) that records in the memory whatever factor is generated by the subexpression $(\mathtt{a}\vee\mathtt{b})^*$ and then recalls the memory after reading d.

If regex are represented as MFA, their structure is much easier to analyse (i. e., in form of a directed, edge-labelled graph). In particular, we can conveniently abstract from the actual backreferences by interpreting an MFA as an NFA that accepts a regular language over the alphabet $\Sigma\cup\{\mathtt{o}(i),\mathtt{c}(i),i \mid 1\leq i\leq k\}$; in this way, a regex even directly translates into a *normal* regular expression. For example, regex s from above can be seen as the regular expression $\mathtt{o}(x)(\mathtt{a}\vee\mathtt{b})^*\mathtt{c}(x)\mathtt{d}x$ that generates words like $\mathtt{o}(x)\,\mathtt{a}\,\mathtt{c}(x)\,\mathtt{d}x$, $\mathtt{o}(x)\,\mathtt{aba}\,\mathtt{c}(x)\,\mathtt{d}x$, $\mathtt{o}(x)\,\mathtt{bbab}\,\mathtt{c}(x)\,\mathtt{d}x$ and so on (obviously, the actual regex language can be obtained from this regular language by replacing variable occurrences x with the corresponding words enclosed in $\mathtt{o}(x)\ldots\mathtt{c}(x)$, which is exactly what an MFA does "on-the-fly" in a computation).

3. Our Contribution

We develop two different approaches to efficient regex matching:

Regex with bounded active variable degree: We define a complexity parameter of regex, called *active variable degree* (denoted by $\mathsf{avd}(\alpha)$). Intuitively speaking, this parameter measures the number of variables that can be *active* at the same time in a match, and the algorithmic application relies on devising a matching procedure, which, in a sense, reuses variables that are currently *not active*.

Theorem 3.1 *Regex with bounded active variable degree can be matched in polynomial-time.*

This approach can also be seen as a technique to reduce the number of variables of a regex, a problem that, in its general form, is undecidable (see [5]).

Memory-deterministic regex: A classical regular expression is called *deterministic* if it directly translates into a deterministic automaton (like $\mathtt{a}(\mathtt{ba})^*$, but unlike $(\mathtt{ab})^*\mathtt{a}$). This classical concept has intensively been researched. Recently, deterministic regular expressions have been extended by backreferences in order to define *deterministic regex* (*det-regex* for short) [6]. For example, $x\{(\mathtt{a}\vee\mathtt{b})^*\}x$ is not deterministic, but $x\{(\mathtt{a}\vee\mathtt{b})^*\}\mathtt{c}x$ is. While det-regex have several desirable properties, they seem unnecessarily restricted if efficient matching is our main concern. For example, every non-deterministic classical regular expression, which can be matched efficiently by standard techniques, is also not a det-regex and, as shown in [6], the class of det-regex does not even cover all regular languages. In fact, any "non-deterministic fragment" like $(\mathtt{ab})^*\mathtt{a}$ immediately makes a regex non-deterministic, even though it might be rather harmelss with respect to the matching complexity. Consider for example the non-deterministic regex $x\{(\mathtt{ab})^*\mathtt{a}\}y\{\mathtt{c}^+x\}y$. In all the non-deterministic computational branches of the corresponding MFA the already consumed prefix of the input and the contents of the memories are always the same, the only difference is whether the last consumed a is the first or the second one of the subexpression $(\mathtt{ab})^*\mathtt{a}$. On the other hand, the non-determinism of $\mathtt{a}^*x\{\mathtt{a}^*\}y\{\mathtt{a}^*\}xy$ is much more problematic: the many different ways of how occurrences of a can be stored in the memories immediately lead to a combinatorial explosion of computational branches (also note that depending on the memory contents, the different computations consume different prefixes of

the remaining input once memories are recalled). Hence, non-determinism seems only problematic with respect to how memories are used, but can be handled if it is somehow limited to just branching the computations into several states.

Formalising this intuitive idea turns out to be quite challenging on a technical level. We define *memory-deterministic regex*, which enforce some synchronisation between the different computational branches of the corresponding MFA.

Theorem 3.2 *Memory-deterministic regex can be matched in time* $\mathrm{O}(|w||r|^3(|\Sigma|+k))$*, where* w *is the word,* r *the regex,* Σ *the alphabet and* k *the number of backreferences. Checking whether a given regex* r *is memory-deterministic can be done in time* $\mathrm{O}(|r|^5)$.

Lower bounds: The active variable degree can be improved to a stronger complexity parameter (that also might be exploited in similar ways), but computing it is coNP-hard. Even rather strong restrictions of non-determinism will lead to intractability, as long as these restrictions are of a local and syntactical nature. This observation leads to a regex-property that is is sufficient for efficient matching, but coNP-hard to be checked for. The concept of memory determinism results from finding a balance between matching- and checking-complexity.

References

[1] A. V. AHO, Algorithms for Finding Patterns in Strings. In: *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*. 1990, 255–300.

[2] D. ANGLUIN, Finding Patterns Common to a Set of Strings. *J. Comput. Syst. Sci.* **21** (1980) 1, 46–62.

[3] H. FERNAU, M. L. SCHMID, Pattern matching with variables: A multivariate complexity analysis. *Information and Computation (I&C)* **242** (2015), 287–305.

[4] H. FERNAU, M. L. SCHMID, Y. VILLANGER, On the Parameterised Complexity of String Morphism Problems. *Theory of Computing Systems (ToCS)* **59** (2016) 1, 24–51.

[5] D. D. FREYDENBERGER, Extended Regular Expressions: Succinctness and Decidability. *Theory of Computing Systems (ToCS)* **53** (2013) 2, 159–193.

[6] D. D. FREYDENBERGER, M. L. SCHMID, Deterministic regular expressions with back-references. *J. Comput. Syst. Sci.* **105** (2019), 1–39.
`https://doi.org/10.1016/j.jcss.2019.04.001`

[7] S. KLEENE, Representation of events in nerve nets and finite automata. In: C. SHANNON, J. MCCARTHY (eds.), *Automata Studies*. Annals of Mathematics Studies 34, Princeton University Press, 1956, 3–41.

[8] F. MANEA, M. L. SCHMID, Matching Patterns with Variables. *CoRR* **abs/1906.06965** (2019).
`http://arxiv.org/abs/1906.06965`

[9] D. REIDENBACH, M. L. SCHMID, Patterns with bounded treewidth. *Information and Computation (I&C)* **239** (2014), 87–99.

[10] M. L. SCHMID, Characterising REGEX languages by regular languages equipped with factor-referencing. *Information and Computation (I&C)* **249** (2016), 1–17.